WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Tuesday, March 16, 2004

Hide?	Set Name	Query	Hit Count
		DB=USOC,EPAB,JPAB,DWPI,TDBD; PLUR=NO; OP=ADJ	
☐	L17	predicate$1 and (reservation station$1)	3
☐	L16	predicate$1 and (reservation station$1) and scoreboard$1	0
		DB=PGPB,USPT; PLUR=NO; OP=ADJ	
☐	L15	5892936.pn. and predicate$1	1
☐	L14	6513109.pn. and reorder and reservation	1
☐	L13	6513109.pn. and reorder	1
☐	L12	L10 or L11	18
☐	L11	L6 and re-order buffer$1	5
☐	L10	L6 and reorder buffer$1	17
☐	L9	L6 and scoreboard$1	5
☐	L8	L7 and scoreboard$1	1
☐	L7	predicate$1 same reservation station$1	4
☐	L6	predicate$1 and reservation station$1	23
☐	L5	predicate and (reservation station)	10
☐	L4	predicate slip	1
☐	L3	predicate$1 same (reservation station$1) and scoreboard$1	1
		DB=USOC,EPAB,JPAB,DWPI,TDBD; PLUR=NO; OP=ADJ	
☐	L2	stall$3 with predicated instruction$1	1
		DB=PGPB,USPT; PLUR=NO; OP=ADJ	
☐	L1	stall$3 with predicated instruction$1	1

END OF SEARCH HISTORY

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Tuesday, March 16, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT; PLUR=NO; OP=ADJ	
☐	L14	5892936.pn. and predicate$1	1
☐	L13	5892936.pn. am	0
☐	L12	6513109.pn. and reorder and reservation	1
☐	L11	6513109.pn. and reorder	1
☐	L10	l8 or L9	18
☐	L9	L4 and re-order buffer$1	5
☐	L8	L4 and reorder buffer$1	17
☐	L7	L4 and scoreboard$1	5
☐	L6	L5 and scoreboard$1	1
☐	L5	predicate$1 same reservation station$1	4
☐	L4	predicate$1 and reservation station$1	23
☐	L3	predicate and (reservation station)	10
☐	L2	predicate slip	1
☐	L1	predicate$1 same (reservation station$1) and scoreboard$1	1

END OF SEARCH HISTORY

WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Tuesday, March 16, 2004

Hide?	Set Name	Query	Hit Count
	DB=USOC; PLUR=NO; OP=ADJ		
☐	L8	register$1 same L7	26
☐	L7	scoreboard	180
☐	L6	renam$3 with scoreboard	0
	DB=PGPB,USPT; PLUR=NO; OP=ADJ		
☐	L5	renam$3 with scoreboard	44
☐	L4	slip$4 with predicate$1	18
☐	L3	predicated dependencies	3
☐	L2	non-predicated dependencies	1
☐	L1	reservation station and predicate and reorder buffer	6

END OF SEARCH HISTORY

First Hit

☐ Generate Collection | Print

L5: Entry 1 of 4 File: PGPB Aug 15, 2002

PGPUB-DOCUMENT-NUMBER: 20020112148
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020112148 A1

TITLE: System and method for executing predicated code out of order

PUBLICATION-DATE: August 15, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Wang, Perry	San Jose	CA	US	
Wang, Hong	Fremont	CA	US	
Kling, Ralph	Sunnyvale	CA	US	
Ramakrishnan, Kalpana	Sarataoga	CA	US	

APPL-NO: 09/ 737783 [PALM]
DATE FILED: December 15, 2000

INT-CL: [07] G06 F 9/00

US-CL-PUBLISHED: 712/226
US-CL-CURRENT: 712/226

REPRESENTATIVE-FIGURES: 7A

ABSTRACT:

According to one aspect of the present invention, a system including a pipeline microprocessor for out-of-order processing of predicated instructions is disclosed. The microprocessor includes multiple dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes at least one guarding predicate. The microprocessor also includes a register renaming unit, a reorder buffer, multiple execution units and multiple reservation stations. The register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the dynamic pipeline stages. The microprocessor also includes an augmented register alias table. Also disclosed is a method of operating a microprocessor for out-of-order processing of predicated instructions.

First Hit

Generate Collection | Print

L5: Entry 1 of 4 File: PGPB Aug 15, 2002

DOCUMENT-IDENTIFIER: US 20020112148 A1
TITLE: System and method for executing predicated code out of order

Abstract Paragraph:
According to one aspect of the present invention, a system including a pipeline microprocessor for out-of-order processing of predicated instructions is disclosed. The microprocessor includes multiple dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes at least one guarding predicate. The microprocessor also includes a register renaming unit, a reorder buffer, multiple execution units and multiple reservation stations. The register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the dynamic pipeline stages. The microprocessor also includes an augmented register alias table. Also disclosed is a method of operating a microprocessor for out-of-order processing of predicated instructions.

Detail Description Paragraph:
[0024] As will be described in more detail below, one embodiment includes a system including a pipeline microprocessor for out-of-order processing of predicated instructions is disclosed. The microprocessor includes multiple dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes at least one guarding predicate. The microprocessor also includes a register renaming unit, a reorder buffer, multiple execution units and multiple reservation stations. The register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the dynamic pipeline stages. The microprocessor also includes an augmented register alias table. Also disclosed is a method of operating a microprocessor for out-of-order processing of predicated instructions.

Detail Description Paragraph:
[0027] Conventional dynamic execution microarchitectures use reservation stations 130, to remove issue blockages due to pending data dependencies in predicate-free code. To similarly execute predicated code without introducing any additional or special hardware, the baseline performance embodiment treats the guarding predicate of an instruction as one of the source operands.

Detail Description Paragraph:
[0029] A first problem occurs during scheduling steps 158, 159 when a predicated instruction continuously waits in the reservation stations 130 for the predicate-defining instruction to finish. A second problem arises at the rename stage 155, 156 before the instructions enter the dynamic portion of the processor. With multiple definitions assigned to a common register, which is guarded by different predicates, the renaming mechanism may need to stall when the predicates are not resolved. As a result, "bubbles" or stalls can be introduced in the pipeline.

Detail Description Paragraph:
[0030] For the baseline performance embodiment described above, when a predicate has not yet been produced, all instructions that depend on this predicate must wait in the reservation stations 130. Even if all the other source operands are available, the instruction cannot be executed until the predicate is ready. In

situations where some predicates have not been resolved, the reservation stations 130 will start to pile up with those instructions having unresolved guarding predicates. As a result, the reservation stations 130 can become saturated quickly and induce backpressure on the pipeline. In other words, because of the unresolved predicates, the pipeline may stall due to the saturation of reservation station 130 entries, thereby causing performance losses.

Detail Description Paragraph:
[0044] Register R is renamed to different physical registers as R's defining instructions enter the rename stage. The physical identifiers are recorded by the renaming mechanism. When the select-.mu.op is to be generated, the recorded identifiers are copied to the source operands of the select-.mu.op. The s0 operand is copied with the physical identifier defined by the first instruction. The rest of one, two, or three physical identifiers fill the source operands in the order from s1 to s3. The processor then allocates a new physical register and assigns it to the destination (dest) operand. Thus, this format handles at most three parallel predicated instructions writing to the same register. Therefore, any of the four source operands is a candidate that potentially holds the final value, and the destination operand is where the final value is assigned. Once the select-.mu.op is formed, the processor inserts the select-.mu.op with the in-flight instructions and loads the select-.mu.op into the reservation station. The renaming unit, which does not need to wait for the resolution of the select-.mu.op, can then rename the subsequent uses of register R to the destination register of the select-.mu.op. The priority information of the source operands is inherent in the select-.mu.op, with s3 representing the highest priority. When the status bits of s3 indicate the operand is valid and ready, the select-.mu.op can immediately be executed without waiting on the resolution of the rest of the source operands. For one embodiment, the priority of the source operands is laid out, from left to right, in the program order that the instructions are fetched. Thus, the youngest defining instruction always has the highest priority.

Detail Description Paragraph:
[0058] For one embodiment of a dynamic execution model, the reservation station receives two bits of bypassed information for the status bits of the source operands in a select-.mu.op. One bit (bit1) signals that the computation of the operand has completed and the bypassed data is ready. Bit1 corresponds to the v-bit of the source operand. The other bit (bit2) indicates whether the bypassed data is to be committed or discarded, which is equivalent to the predicate of the result-producing instruction. Bit2 corresponds to the p-bit of the source operand. The status bits, v-bit and p-bit, in the select-.mu.op determine the select-.mu.op dispatch policy. One embodiment of the logic 900 that realizes the dispatching condition with the source fan-in of 4 is shown in FIG. 9. When the highest priority operand (s3) is available, v3 becomes 1. Depending on p3, which is the predicate value, the select-.mu.op can be immediately dispatched if p3 is 1. If p3 is 0, the select-.mu.op must wait for the select-.mu.op's lower priority operands to become available.

Detail Description Paragraph:
[0068] After all registers have been renamed, the predicated register alias table (RAT) detects the renaming of r40, and dynamically attaches select-.mu.ops with the instruction bundle. Once the select-.mu.ops have been injected, the instructions enter the issue stage for dispersal. The issue unit disperses the instructions to several independent reservation stations. For one embodiment, the processor has a centralized reservation station dispatching instructions to two Integer functional units (I-unit) and two Memory functional units (M-unit). The reservation stations can dispatch any instruction when all except predicate dependencies are satisfied. The reason, as we previously mentioned, is that we can slip the predicated instructions and not commit their results until later when the predicate is known. We also assume that all integer operations take 1 cycle and load instructions 2 cycles. Since this paper does not deal with the dispersal rules of the issue unit,

we simply assume a greedy algorithm that issues up to 4 instructions per cycles. FIGS. 12A-G illustrate benefits of select-.mu.op dynamic execution on the right side 1205 of each figure. Static execution is illustrated on the left side 1210 of each figure for comparison.

CLAIMS:

1. A microprocessor comprising: a plurality of dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes a plurality of guarding predicates; a register renaming unit; a reorder buffer; a plurality of execution units; a plurality of reservation stations wherein the register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the plurality of dynamic pipeline stages; and an augmented register alias table.

14. A computer system comprising: a processor, wherein the processor includes: a plurality of dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes a plurality of guarding predicates; a register renaming unit; a reorder buffer; a plurality of execution units; a plurality of reservation stations wherein the register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the plurality of dynamic pipeline stages; and an augmented register alias table; a system bus; a computer memory system; an input/output device; wherein the system bus is coupled to the processor, the computer memory system and the input/output device.

First Hit Fwd Refs

Generate Collection Print

L5: Entry 3 of 4 File: USPT Dec 28, 1999

US-PAT-NO: 6009512
DOCUMENT-IDENTIFIER: US 6009512 A

TITLE: Mechanism for forwarding operands based on predicated instructions

DATE-ISSUED: December 28, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Christie; David S.	Austin	TX		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 08/ 958477 [PALM]
DATE FILED: October 27, 1997

INT-CL: [06] G06 F 9/318

US-CL-ISSUED: 712/226; 712/218
US-CL-CURRENT: 712/226; 712/218

FIELD-OF-SEARCH: 395/582, 395/581, 395/583, 395/567, 395/394, 712/235, 712/234, 712/236, 712/226, 712/218

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5471593	November 1995	Branigin	712/235

OTHER PUBLICATIONS

Mahlke, Scott A. et al., A Comparison of Full and Partial Predicated Execution Support for ILP Processors, 1995, ACM, pp. 138-149.
Johnson, Richard et al., Analysis Techniques for Predicated Code, 1996, IEEE, pp. 100-113.
Tyson, Gary Scott, The Effects of Predicated Execution on Branch Prediction, 1994, ACM, pp. 196-206.
August, David J. et al., A Framework for Balancing Control Flow and Predication, 1997, IEEE, pp. 92-103.

Kathail et al., HPL Playdoh Architecture Specification: Version 1.0, Hewlett Packard, Computer Systems Laboratory, HPL-93-80, Feb., 1994, pp. 1-48.

ART-UNIT: 273

PRIMARY-EXAMINER: Ellis; Richard L.

ATTY-AGENT-FIRM: Conley, Rose & Tayon Merkel; Lawrence J. Kivlin; B. Noel

ABSTRACT:

A method and apparatus for providing predicated instructions in a processor employing out of order execution. In one embodiment, a plurality of decode units are configured to decode a plurality of variable byte length instructions and to provide a plurality of output of signals. The output signals are provided to a plurality of reservation stations coupled to the plurality of decode units within the superscalar microprocessor. Functional units are configured to receive the output signals from the plurality of decode units. The functional units include function execution units coupled to receive signals from the plurality of reservation stations and to provide a function output responsive to the output signals. The functional units further comprise a predication unit configured to determine whether a predetermined condition has occurred and either stop the function output or allow the function output to be transmitted depending on whether the predetermined condition has occurred.

22 Claims, 14 Drawing figures

First Hit Fwd Refs

Generate Collection Print

L5: Entry 3 of 4 File: USPT Dec 28, 1999

DOCUMENT-IDENTIFIER: US 6009512 A
TITLE: Mechanism for forwarding operands based on predicated instructions

Abstract Text (1):
A method and apparatus for providing predicated instructions in a processor employing out of order execution. In one embodiment, a plurality of decode units are configured to decode a plurality of variable byte length instructions and to provide a plurality of output of signals. The output signals are provided to a plurality of reservation stations coupled to the plurality of decode units within the superscalar microprocessor. Functional units are configured to receive the output signals from the plurality of decode units. The functional units include function execution units coupled to receive signals from the plurality of reservation stations and to provide a function output responsive to the output signals. The functional units further comprise a predication unit configured to determine whether a predetermined condition has occurred and either stop the function output or allow the function output to be transmitted depending on whether the predetermined condition has occurred.

Brief Summary Text (26):
Accordingly, a method and apparatus for providing predicated instructions in a processor employing out of order execution is provided herein. In one embodiment, a plurality of decode units are configured to decode a plurality of variable byte length instructions and to provide a plurality of output of signals. The output signals are provided to a plurality of reservation stations coupled to the plurality of decode units within the superscalar microprocessor. Functional units are configured to receive the output signals from the plurality of decode units. The functional units include function execution units coupled to receive signals from the plurality of reservation stations and to provide a function output responsive to the output signals. The functional units further comprise a predication unit configured to determine whether a predetermined condition has occurred and either stop the function output or allow the function output to be transmitted depending on whether the predetermined condition has occurred.

Detailed Description Text (6):
Referring next to FIG. 5, a block diagram of a microprocessor 1200 including a predicated instruction mechanism is shown. As illustrated in FIG. 5, superscalar microprocessor 1200 (preferably an x86-type processor) includes a prefetch/predecode unit 1202 and a branch prediction unit 1120 coupled to an instruction cache 1204. A main memory, typically separate from processor 1200, is coupled to processor 1200 by way of prefetch/predecode unit 1202. Instruction alignment unit 1206 is coupled between instruction cache 1204 and a plurality of decode units 1208A-1208C (referred to collectively as decode units 1208). Each decode unit 1208A-1208C is coupled to a respective reservation station units 1210A-1210C (referred to collectively as reservation stations 1210), and each reservation station 1210A-1210C is coupled to a respective functional unit 1212A-1212C (referred to collectively as functional units 1212). Decode units 1208, reservation stations 1210, and functional units 1212 are further coupled to a reorder buffer 1216, a register file 1218 and a load/store unit 1222. A data cache 1224 is finally shown coupled to load/store unit 1222, and an MROM unit 1209 is shown coupled to instruction alignment unit 1206 and decode units 1208. Each functional unit 1212

includes a predication unit 1213A-1213C (referred to collectively as predication unit 1213), whose use will be described in greater detail below.

Detailed Description Text (12):
Before proceeding with a detailed description of the predicated instruction mechanism, general aspects regarding other subsystems employed within the exemplary superscalar microprocessor 1200 of FIG. 5 will be described. For the embodiment of FIG. 5, each of the decode units 1208 includes decoding circuitry for decoding the predetermined fast path instructions referred to above. In addition, each decode unit 1208A-1208F routes displacement and immediate data to a corresponding reservation station unit 1210A-1210F. Output signals from the decode units 1208 include bit-encoded execution instructions for the functional units 1212 as well as operand address information, immediate data and/or displacement data.

Detailed Description Text (25):
Turning now to FIG. 6, there is shown a predication mechanism for use in a processor such as processor 1200 implementing predicate execution and out of order instruction execution. Processor 1200 is, for example, an X86 compatible processor in which outstanding dependencies are resolved, as discussed above, by forwarding results into reservation station 102. Reservation station 102 may, for example, be reservation station 1210A, 1210B, or 1210C as shown in FIG. 5. An instruction that references a register that is the target of a predicated instruction has one of two possible dependencies: Either on the result of the predicated instruction if its condition for execution is met, or on the result of whichever prior instruction to the predicated instruction last updated the register. Thus, the predicated instructions are forwarded the original contents of the destination register as an input operand. FIG. 6 illustrates reservation station 102, which includes operand A 108, operand B 110, and predicated instruction 106. Additionally, FIG. 6 illustrates a functional unit 120, which includes predication unit 117 and execution unit 114. Functional unit 120 may, for example, be functional unit 1212A, 1212B, or 1212C. At the same time that the operation is performed the condition code or predicate flag is evaluated in evaluation unit 116, as will be discussed in more detail below. The result of the evaluation is used to control a multiplexer 118, which receives the result of execution unit 114 and the output from operand A 108 which stores the original contents of the destination operand. Multiplexer 118 selects either the result of the operation, if the predicate condition is true, or the input operand that is the original value of the destination, if the predicate condition is false. Multiplexer 118 is also coupled to functional unit 120. Further dependencies are handled by making subsequent instructions unconditionally dependent on the result of the predicated instruction without concern for the condition code or predicate flag.

First Hit Fwd Refs

Generate Collection Print

L7: Entry 3 of 5 File: USPT Nov 9, 1999

US-PAT-NO: 5983335
DOCUMENT-IDENTIFIER: US 5983335 A
** See image for Certificate of Correction **

TITLE: Computer system having organization for multiple condition code setting and for testing instruction out-of-order

DATE-ISSUED: November 9, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Dwyer, III; Harry	Annandale	NJ		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 08/ 841550 [PALM]
DATE FILED: April 30, 1997

PARENT-CASE:
This application is a division of U.S. patent application Ser. No. 08/328,933 filed Oct. 25, 1994, now U.S. Pat. No. 5,630,157 which is a continuation of Ser. No. 07/721,853 filed Jun. 13, 1991 and now abandoned.

INT-CL: [06] G06 F 15/00

US-CL-ISSUED: 712/23; 712/217, 712/216, 712/215
US-CL-CURRENT: 712/23; 712/215, 712/216, 712/217

FIELD-OF-SEARCH: 395/800.23, 395/800.24, 395/390, 395/391, 395/392, 395/393, 395/566, 711/125, 711/149, 711/217, 712/23, 712/24, 712/225, 712/214, 712/215, 712/216

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4293910	October 1981	Flusche et al.	
☐	4498136	February 1985	Sproul, III	

☐	4654785	March 1987	Nishiyama et al.	
☐	4722049	January 1988	Lahti	
☐	4760520	July 1988	Shintani et al.	
☐	4807115	February 1989	Horng	
☐	4833599	May 1989	Colwell et al.	
☐	4853840	August 1989	Shibuya	
☐	4858105	August 1989	Kuriyama et al.	395/800.23
☐	4901233	February 1990	Liptay	
☐	4903196	February 1990	Pomerene et al.	
☐	4942525	July 1990	Shintani et al.	
☐	4972342	November 1990	Davis et al.	
☐	4991080	February 1991	Emma et al.	
☐	4991090	February 1991	Emma et al.	
☐	4991133	February 1991	Davis et al.	
☐	5003462	March 1991	Blaner et al.	
☐	5072364	December 1991	Jardine et al.	
☐	5075844	December 1991	Jardine et al.	
☐	5142634	August 1992	Fite et al.	
☐	5185868	February 1993	Tran	
☐	5185871	February 1993	Frey et al.	
☐	5186697	February 1993	Johnson	
☐	5197132	March 1993	Steely, Jr. et al.	
☐	5226126	July 1993	McFarland et al.	
☐	5226130	July 1993	Favor et al.	
☐	5228131	July 1993	Ueda et al.	
☐	5257354	October 1993	Comfort et al.	
☐	5265213	November 1993	Weiser et al.	
☐	5283873	February 1994	Steely, Jr. et al.	
☐	5287467	February 1994	Blaner et al.	
☐	5428811	June 1995	Hinton et al.	395/800.23
☐	5471593	November 1995	Brangin	
☐	5488729	January 1996	Vegesna et al.	395/800.23

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0301220	February 1989	EP	
0340453	November 1989	EP	

0378830	July 1990	EP
0381471	August 1990	EP

OTHER PUBLICATIONS

"Design of a Computer: The Control Data 6600"; J.E. Thornton; Scott, Foreman & Company; Glenview, IL; 1970; pp. 12-140.
"Detection and Parallel Execution of Independent Instructions"; Garold S. Tjaden et al.; IEEE Transactions on Computers, vol. C-19, No. 10, Oct. 1970; pp. 889-895.
"The IBM System/360 Model 91: Storage System"; L.J. Boland et al.; IBM Journal, Jan. 1967; pp. 54-68.
"A Fast Instruction Dispatch Unit for Multiple and Out-of-Sequence Issuances"; H.C. Torng; Professor of Elec. Engineering; Stanford University; Nov. 1987.
"Limits on Multiple Instruction Issue"; Michael D. Smith et al.; Stanford University; ASPLOS-III Proceedings; Apr. 1989.
"Design Coices for the HPSm Microprocessor Chip"; Wen-mei Hwu et al.; Proceedings of the 20th Annual Hawaii International Conference on System Sciences; 1987; pp. 330-336.
"Reducing the Cost of Branches"; Scott McFarling et al.; Computer Systems Laboratory; Stanford University; 1986 IEEE; pp. 396-403.
"The ZS-1 Central Processor"; J.E. Smith et al.; Astronautics Corporation os America; Madison, Wisconsin; 1987 ACM; pp. 199-204.
"Designing a VAX for High Performance"; Tryggve Fossum et al.; 1990 IEEE; pp. 36-43.
"Pre-Decoding Mechanism for Superscalar Architecture"; Kenji Minagawa et al.; Toshiba Research and Development Center; Japan; 1991 IEEE; pp. 21-24.
"Fast Reset of Logical Facilities"; IBM Technical Disclosure Bulletin, vol. 31, No. 6, Nov. 1988; p. 275.
"Pipelined Register-Storage Architectures"; Steven R. Kunkel et al.; University of Wisconsin-Madison; 1986 IEEE; pp. 515-517.
"An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors"; Ramon D. Acosta et al.; 1986 IEEE; pp. 815-828.
Instruction Issue Logic for High-Performance, Interruptable Pipelined Processor; Gurindar S. Sohi et al.; University of Wisconsin-Madison; 1987 ACM.
"Implementing Precise Interrupts in Pipelined Processors"; James E. Smith; IEEE, vol. 37, No. 5, May 1988; pp. 562-573.
"The Metaflow Architecture"; Val Popescu et al.; IEEE Micro; 1991 IEEE; Jun. 1991; vol. 11, No. 3; pp. 10-73.
"The Performance Potential of Multiple Functional Unit Processors"; A.R. Pleszkun et al.; University of Wisconsin-Madison; 1988 IEEE; Computer Architecture News; vol. 16, No. 2; pp. 37-44.
"Cache Memories"; Alan Jay Smith; Computing Surveys; vol. 14, No. 3, Sep. 1982; pp. 473-530.
"Dhrystone: A Synthetic Systems Programming Benchmark"; Reinhold P. Weicker; Computing Practices; Oct. 1984, vol. 27, No. 10; pp. 1013-1030.
"VSLI Processor Architecture"; John L. Hennessy; IEEE Transactions on Computers; vol. C-33, No. 12, Dec. 1984; pp. 1221-1246.
"Concurrent VLSI Architectures"; Charles L. Seitz; IEEE Transactions on Computers;, vol. C-33, No. 12, Dec. 1984; pp. 1246-1265.
"Critical Issues Regarding HPS, A High Performance Microarchitecture"; Yale N. Patt et al.; University of California, Berkley; 1985 ACM; pp. 109-116.
"HPSm, a High Performance Restricted Data Flow Architecture Having Minimal Functionality"; Wen-mei Hwu et al.; University of California, Berkley; IEE 1986.
"New Computers for Artificial Intelligence Processing"; Benjamin W. Wah; University of Illinois at Urbana-Champaign; 1987 IEEE; pp. 10-15.
"Reducing the Branch Penalty in Pipelined Processors"; David J. Lilja; University of Illinois at Urbana-Champaign; 1988 IEEE; pp. 47-55.
"A VLIW Architecture for a Trace Scheduling Compiler"; Robert P. Colwell et al.; IEEE Transactions on Computers, vol. 37, No. 8, Aug. 1988; pp. 967-979.

"HPSm2; A Refined Single-Chip Microengine"; Wen-mei W. Hwu et al.; University of Illiniois, Urbana; 1988 IEEE; pp. 30-40.
"The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance"; Norman P. Jouppi; 1989 IEEE; pp. 1645-1658.
"Building Parallelism Into the Insruction Pipeline"; Scott Chan et al.; Tandem Computers, Inc., Cupertino, California; High Performance Systems; pp. 52-58.
"SIMP: A Novel High-Speed Single-Processor Architecture"; Kazuaki Murakami et al.; Dept. of Information Systems; Kyushu University, Fukuoka, Japan; 1989 ACM; pp. 78-85.
"Available Instruction-Level Parallelism for Superscalar and Superpiplined Machines"; Norman P. Jouppi et al. Digital Equipment Corp.; Western Research Lab; 1989 ACM; pp. 272-282.
"Instruction Schuduling for the IBM RISC System/6000 Processor"; H.S. Warren, Jr.; IBM J. Res. Develop., vol. 34, No. 1, Jan. 1990.
"Machine Organization of the IBM RISC System/6000 Processor"; G.F. Grohoski; IBM J. Res. Develop., vol. 34, No. 1, Jan. 1990; pp. 37-58.
"Interrupt Handling for Out-of-Order Execution Processors"; H.C. Torng et al.; School of Elec. Engineering; Phillips Hall, Cornell University, Ithaca, NY; pp. 1-16 (with figures 1-9 included in addition to text pages).
"Limits on Multiple Instruction Issue"; Michael D. Smith et al.; Stanford University; Center for Integrated Systems; 1989 ACM; pp. 290-302.
"Boosting Beyond Static Scheduling in a Superscalar Processor"; Michael D. Smith et al.; Stanford University; Computer Systems Lab; 1990 IEEE; pp. 344-354.
"Dynamic Instruction Scheduling and the Astronautics ZS-1"; James E. Smith; Astronautics Corp. of America; Jul. 1989 IEEE; pp. 21-35.
"Look-Ahead Processors"; Robert M. Keller; Dept. of Elec. Engineering, Princeton University; Computing Surveys, vol. 7, No. 4, Dec. 1975; pp. 177-195.
"Instruction Issue Logic in Pipelined Supercomputers"; Shlomo Weiss et al.; IEEE Transactions on Computers, vol. C-33, No. 11, Nov. 1984; pp. 1013-0122.
"An Instruction Issuing Mechanism for Performance Enhancements"; H.C. Torng; Technical Report; Feb. 1984.
"Branch Prediction Strategies and Branch Target Buffer Design"; Johnny K.F. Lee; Hewlett-Packard; Alan Jay Smith, University of California, Berkeley; Computer, vol. 17, No. 1; Jan. 1984.
"The 801 Minicomputer"; George Radin; IBM J. Res. Develop., vol. 27, No. 3, May 1983; pp. 237-246.
"An Efficient Algorithm for Exploiting Multiple Arithmetic Units"; R.M. Tomasulo; IBM; pp. 293-305 1982.
"A VLSI RISC"; David A. Patterson et al., University of California, Berkeley; 1982 IEEE.
"Checkpoint Repair for High-Performance Out-of-Order Execution Machines"; Wen-Mei W. Hwu et al.; IEEE Transations on Computers, vol. C-36, No. 12; Dec. 1987; pp. 1497-1514.
Acosta, "Evaluation, Implementation, and Enhancement of Dispatch Stack Instruction Issuing Mechanism", Aug. 1995.

ART-UNIT: 273

PRIMARY-EXAMINER: Donaghue; Larry D.

ATTY-AGENT-FIRM: Whitham, Curtis & Whitham Samodovitz; Arthur J.

ABSTRACT:

Computer system with multiple, out-of-order, instruction issuing system suitable for superscalar processors with a RISC organization, also has a Fast Dispatch Stack (FDS), a dynamic instruction scheduling system that may issue multiple, out-of-order, instructions each cycle to functional units as dependencies allow. The basic issuing mechanism supports a short cycle time and its capabilities are augmented.

Condition code dependent instructions issue in multiples and out-of-order. A fast register renaming scheme is presented. An instruction squashing technique enables fast precise interrupts and branch prediction. Instructions preceding and following one or more predicted conditional branch instructions may issue out-of-order and concurrently. The effects of executed instructions following an incorrectly predicted branch instruction or an instruction that causes a precise interrupt are undone in one machine cycle.

17 Claims, 72 Drawing figures

L7: Entry 3 of 5 File: USPT Nov 9, 1999

DOCUMENT-IDENTIFIER: US 5983335 A
** See image for Certificate of Correction **
TITLE: Computer system having organization for multiple condition code setting and for testing instruction out-of-order

Drawing Description Text (5):
FIG. 4 shows Thornton's scoreboard system.

Detailed Description Text (68):
2.1 THORNTON'S SCOREBOARD

Detailed Description Text (70):
An instruction is fetched from memory 405 into the instruction stack 410, a set of registers that also holds some previously issued instructions. When a loop is encountered in the instruction stream, a recently executed instruction may often be accessed from the instruction stack instead of memory. The instruction is transferred to a series of instruction registers U.sup.0, U.sup.1, and .andgate.U.sup.2) 415 that decode and analyze it. It is issued from the last of these registers to a functional unit 420. The Unit and Register Reservation Control or scoreboard 425 reserves system resources when the instruction is issued and subsequently controls read operand and store result operations of the functional unit it is issued to. An instruction that cannot issue blocks those behind it. An instruction is issued to a functional unit if the following conditions are met:

Detailed Description Text (74):
2. Functional units inform the scoreboard when results are ready. When WAR hazards have cleared, the scoreboard tells the units to store the results (WAR dependency control). The limitations of this approach are that:

Detailed Description Text (80):
1. Instructions are issued to sets of reservation stations (holding buffers) 505, one set for each functional unit, where they wait until their operands become available. They are then issued to the attached functional unit for execution. If a reservation station is not available for an instruction, it stalls and blocks instructions behind it.

Detailed Description Text (81):
2. A register 510 may contain valid data or the name of a reservation station that will supply it with data in the future.

Detailed Description Text (82):
3. When an instruction is issued, the name of its reservation station is placed into the name field 515 of its destination register. Subsequent issues of instructions using that register as a source will take the name in the name field with them. They now know the name of the reservation station that will generate the data they need.

Detailed Description Text (83):
4. When an instruction completes, its result and reservation station name are

broadcast. All instructions and registers waiting for that result can now identify and copy it.

Detailed Description Text (85):
When an instruction writes to a register, all previously issued instructions that read it have copies of either the data or the reservation station's name that will produce the data. RAW hazards are also eliminated by the sequential nature of instruction issue.

Detailed Description Text (86):
It is interesting that Tomasulo's algorithm can also handle issues of multiple, in-order instructions containing no hazards (although this is not supported in the IBM 360/91). If enough reservation stations are available, the sequence could be issued. The (all different) result registers would not be sourced by other instructions within the sequence. Therefore, the reading of registers and the placing of reservation station names into them would occur correctly. All dependencies between instructions in different blocks would then be handled correctly by the algorithm. Some consider that Tomasulo's algorithm was used in the IBM System/360 model 195 and the IBM System/390.

Detailed Description Text (89):
HPSm is a minimum functionality implementation of High Performance Substrate (HPS) [HwPa 86, 87]. An HPSm instruction contains two operations that may have data dependencies on each other. If they do, the instruction causes the hardware to forward the results of the first operation directly to the second. The execution mechanism examines sequential instructions and decomposes them into two operations, which, after register renaming, are integrated into data structures attached to functional units called node tables. Node tables operate much like Tomasulo's reservation stations. Instructions wait here for operands to be generated by the functional units and are then issued to functional units for execution. FIG. 6 is a block diagram of the HPSm machine.

Detailed Description Text (94):
Sohi improves the efficiency of Tomasulo's algorithm by concentrating all the instruction issuing logic and reservation stations into one mechanism called the Register Update Unit (RUU) [Sohi 90]. A reservation station is no longer permanently coupled to a particular functional unit and may be assigned as needed. As previously noted, Tomasulo's approach will stall if a reservation station is not available on the required functional unit. FIG. 7 shows a diagram of the RUU.

Detailed Description Text (102):
By centralizing data and control, the RUU enhances Tomasulo's approach with improved reservation station utilization and support for precise interrupts. Speedups are reported over serial issue of about 1.5 to 1.7, with one issue unit and bypass logic. Sohi claims that the RUU concept may be expanded to issue multiple instructions. With four issue units and bypass logic, speedups of 1.7 to 1.9 over serial issue are reported [PlSo 88].

Detailed Description Text (211):
The action of a Conditional Branch instruction, q.sub.CB, is predicated on a situation (condition) caused by the execution of one or more preceding instructions. Examples of conditions are an overflow out of a register and a result that is greater than zero. Conditional Branch instructions in different architectures test conditions using a variety of methods.

Detailed Description Text (214):
The data bandwidth of an execution unit is the maximum rate at which storage instructions can transfer data to and from memory. It may be limited to the maximum rate at which storage instructions can be executed (storage instruction throughput) or the maximum rate at which the memory system can store and fetch data (memory

bandwidth). The data bandwidth of an execution unit with multiple FUs has to meet its data requirements; otherwise its throughput will have to be constrained. FUs may idle because the issuance and the execution of instructions dependent on the data are delayed. Previous proposals supporting out-of-order memory accesses schedule at most one memory request per cycle. Thornton's Scoreboard, Tomasulo's Reservation Station system, Sohi's RUU and Hwu and Pratt's HPSm issue at most one memory request per cycle (possibly out-of-order). The scheduling of storage instructions with Torng's Dispatch Stack has not been explored. This is one of the advances that I have made, and my improvements in this regard form part of my preferred embodiment. The SIMP mechanism initiates up to 4 read and 4 write requests per cycle to a Data Cache, but the scheduling algorithm of the storage instructions is not described.

First Hit

Generate Collection | Print

L10: Entry 1 of 18 File: PGPB Feb 19, 2004

DOCUMENT-IDENTIFIER: US 20040034678 A1
TITLE: Efficient circuits for out-of-order microprocessors

Summary of Invention Paragraph:
[0074] Different processors reuse entries in their reordering buffer differently. Those that assign tags serially can use each assigned tag as a direct address into the reordering buffer at which to store the renamed instruction. This is the case in FIG. 9. These processors including the Alpha 21264 write values to a canonical register file by compressing the entries in the reorder buffer so that the instruction in Buffer Entry 0 is always the oldest [8]. When scaled up, the circuitry used in the 21264 for compressing the window requires large area and has long critical-path lengths, however.

Summary of Invention Paragraph:
[0075] When the reorder buffer fills up, some processors exhibit performance anomalies. Some processors empty the reorder buffer, instead of wrapping around, and commit all results to the register file before they start back at the beginning. Some processors wrap around, but start scheduling newer instructions instead of older ones, which can hurt performance. (See [33] for an example of a circuit that works that way.)

Detail Description Paragraph:
[0282] On every clock cycle, a superscalar processor allocates a tag to the result register of every instruction entering the rename stage. Different superscalar implementations allocate tags differently. For example, some processors assign tags in a forward or wrap-around sequence. That is, the first instruction uses Tag 0, the next uses Tag 1, until all the tags are used up, by which time some of the early tags have retired, and we go back and start using Tags 0 and 1 again. Often, the tag number is the same as the number of the reservation station. The allocation of tags in such a system is the same as the allocation of reservation stations in the reordering buffer. The disadvantage of such a scheme is that the tags and the buffer entries are allocated together, and the tags cannot be reused without clearing out the buffer entries (e.g., by writing them to a canonical register file.) Such a write has typically been performed by a multiported register file which has poor scaling properties.

Detail Description Paragraph:
[0351] These circuits can be used to implement the scheduling stage of a superscalar processor which selects a set of ready instructions to run on functional units. The instructions are in reservation stations arrayed in the instruction window. Each reservation station contains information indicating whether its instruction is ready to run, and which functional unit is needed. The functional units include floating point adders, multipliers, integer units, and memory units. At the beginning of every clock cycle, some of the instructions in the reservation stations are ready to run. There may be more than one functional unit that can be used by any particular instruction. In many situations, certain of the instructions should be scheduled with higher priority than others. For example, it may be preferred to schedule the oldest instructions with higher priority, or it may be preferred to schedule instructions which resolve condition codes with higher priority. For memory operations it may be preferable to schedule nonspeculative

loads rather than speculative loads.

Detail Description Paragraph:
[0365] FIG. 40 shows a butterfly network implemented with switches 720 of degree two. (See, for example, [15, 20] for methods for routing in this kind of network.) This network can move data to the three functional units, illustratively adders from eight window locations 0-7. Switches 720 are connected in a butterfly network. The addresses are the G.sub.ij, values provided by the scheduler CSPP circuits. The data are the two register values that the reorder buffer provides. Shown is a signal containing the address and a signal containing data from each of the 8 reorder buffer slots.

Detail Description Paragraph:
[0536] It is possible to employ even more speculation. For example, several recent researchers have reported that data speculation schemes often produces the right value. (For example, the value produced by an instruction when it executes is a good predictor of the value that will be produced by that instruction the next time it executes.) The reason this works is that a branch speculation may cause a particular instruction, X, to execute and then produce a value. Later it turns out that the branch speculation was incorrect, and then Instruction X must be executed again. Often Instruction X produces the same value, however. (We have read on Newsgroup comp .arch that as many as 30% of all instructions can be predicted for the Intel x86 architecture. Intel calls this value speculation Instruction Reuse. Other studies have been published as well. Processors that employ predicated instructions may gain less advantage from data speculation since the compiler has a chance to make explicit the independence between the branch and the subsequent instruction. Machines with predicated instructions include the HP PA-RISC, the Advanced Risc Machines ARM, and the proposed Intel 1A64 architecture.)

Detail Description Paragraph:
[0640] [27] Robert W. Martell and Glenn J. Hinton. Method and apparatus for scheduling the dispatch of instructions from a reservation station. U.S. Pat. No. 5,519,864,21 May 1996.

First Hit

☐ Generate Collection Print

L10: Entry 1 of 18 File: PGPB Feb 19, 2004

PGPUB-DOCUMENT-NUMBER: 20040034678
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20040034678 A1

TITLE: Efficient circuits for out-of-order microprocessors

PUBLICATION-DATE: February 19, 2004

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Kuszmaul, Bradley C.	Woodbridge	CT	US	
Henry-Kuszmaul, Dana Sue	Woodbridge	CT	US	

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	COUNTRY	TYPE CODE
Yale University				02

APPL-NO: 10/ 608621 [PALM]
DATE FILED: June 27, 2003

RELATED-US-APPL-DATA:
Application 10/608621 is a continuation-of US application 09/267827, filed March 12, 1999, US Patent No. 6609189
Application is a non-provisional-of-provisional application 60/077669, filed March 12, 1998,
Application is a non-provisional-of-provisional application 60/108318, filed November 13, 1998,

INT-CL: [07] G06 F 7/38

US-CL-PUBLISHED: 708/446
US-CL-CURRENT: 708/446

REPRESENTATIVE-FIGURES: 13

ABSTRACT:

The poor scalability of existing superscalar processors has been of great concern to the computer engineering community. In particular, the critical-path delays of many components in existing implementations grow quadratically with the issue width and the window size. This patent presents a novel way to reimplement these components and reduce their critical-path delay growth. It then describes an entire processor microarchitecture, called the Ultrascalar processor, that has better critical-path delay growth than existing superscalars. Most of our scalable designs are based on a single circuit, a cyclic segmented parallel prefix (cspp). We observe that processor components typically operate on a wrap-around sequence of

instructions, computing some associative property of that sequence. For example, to assign an ALU to the oldest requesting instruction, each instruction in the instruction sequence must be told whether any preceding instructions are requesting an ALU. Similarly, to read an argument register, an instruction must somehow communicate with the most recent preceding instruction that wrote that register. A cspp circuit can implement such functions by computing for each instruction within a wraparound instruction sequence the accumulative result of applying some associative operator to all the preceding instructions. A cspp circuit has a critical path gate delay logarithmic in the length of the instruction sequence. Depending on its associative operation and its layout, a cspp circuit can have a critical path wire delay sublinear in the length of the instruction sequence.

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of Provisional Application Serial No. 60/077,669, filed Mar. 12, 1998, and Provisional Application Serial No. 60/108,318, filed Nov. 13, 1998, both of which are incorporated herein by reference in their entireties.

First Hit

Generate Collection | Print

L10: Entry 2 of 18 File: PGPB Aug 15, 2002

DOCUMENT-IDENTIFIER: US 20020112148 A1
TITLE: System and method for executing predicated code out of order

Abstract Paragraph:
According to one aspect of the present invention, a system including a pipeline microprocessor for out-of-order processing of predicated instructions is disclosed. The microprocessor includes multiple dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes at least one guarding predicate. The microprocessor also includes a register renaming unit, a reorder buffer, multiple execution units and multiple reservation stations. The register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the dynamic pipeline stages. The microprocessor also includes an augmented register alias table. Also disclosed is a method of operating a microprocessor for out-of-order processing of predicated instructions.

Summary of Invention Paragraph:
[0001] The present invention relates to computer systems and more specifically relates to in-order microprocessors using predicated instructions.

Summary of Invention Paragraph:
[0003] One approach to improved cooperation between compiler and micro-architecture is using predicated instructions of a predicated execution model.

Summary of Invention Paragraph:
[0004] A predicated execution model is an architectural model where an instruction is guarded by a Boolean operand whose value determines if the instruction is executed or nullified. To explore ILP, a compiler can take full advantage of the predicated execution model by applying a technique referred to as if-conversion. In short, if-conversion is an optimization that converts control flow dependence into data flow dependence. With if-conversion, the compiler can collapse multiple control flow paths and schedule them based only on data dependencies. Even though a predicated execution model exposes more ILP, such a predicated execution model may not always yield enhanced performance. On the compiler side, the predicated execution model requires a detailed analysis of the dynamic behavior of the code and the dynamic resource availability. Since the effectiveness of predication depends on resource availability, the scalability for and compatibility with future-generation machines are important issues to consider. Given the availability of increasing transistor budgets, increasingly more advanced microarchitecture mechanisms can be incorporated. Furthermore, the legacy base of predicated code should be able to continue to perform well on future processor generations.

Brief Description of Drawings Paragraph:
[0012] FIG. 2A shows a predicate status testing flowchart of one embodiment.

Brief Description of Drawings Paragraph:
[0018] FIG. 7A shows a flowchart of one embodiment of a method of processing a predicated instruction.

Brief Description of Drawings Paragraph:

[0019] FIG. 8 illustrates one embodiment of an augmented register alias table (RAT) with predicates.

Detail Description Paragraph:
[0024] As will be described in more detail below, one embodiment includes a system including a pipeline microprocessor for out-of-order processing of predicated instructions is disclosed. The microprocessor includes multiple dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes at least one guarding predicate. The microprocessor also includes a register renaming unit, a reorder buffer, multiple execution units and multiple reservation stations. The register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the dynamic pipeline stages. The microprocessor also includes an augmented register alias table. Also disclosed is a method of operating a microprocessor for out-of-order processing of predicated instructions.

Detail Description Paragraph:
[0025] There are several types and variations of an out of order or dynamic execution processors. A dynamic microarchitecture as a baseline performance embodiment is shown in FIG. 1. The baseline performance embodiment includes a dynamic portion 105 of the processor 100 including a register renaming unit 110, which maps between temporary and architectural files, a reorder buffer 120, a plurality of reservation stations 130, and a plurality of execution units 140. A bus 115 couples the register renaming unit 110, the reorder buffer 120, the plurality of reservation stations 130 and the plurality of execution units 140 together and to the remaining portions of the microprocessor which are not shown. The pipeline shown in FIG. 1A has 15 stages, with 7 stages 155-161 devoted to the dynamic portion 105 of the processor 100. The dynamic pipeline 155-161 begins with a 2-stage rename 155-156, followed by a register read stage 157, a 2-stage schedule 158-159, an execute stage 150, and finally a retire stage 161. In the schedule stage 158-159, the instructions wait in the reservation stations 130 until the data of the source operands become available. After the data from the source operands are loaded into the register, the instruction enters the execute stage 150. In the final retire stage 161, the instructions are retired in order from the reorder buffer.

Detail Description Paragraph:
[0027] Conventional dynamic execution microarchitectures use reservation stations 130, to remove issue blockages due to pending data dependencies in predicate-free code. To similarly execute predicated code without introducing any additional or special hardware, the baseline performance embodiment treats the guarding predicate of an instruction as one of the source operands.

Detail Description Paragraph:
[0028] The baseline performance embodiment poses two performance limitations due to a substantial penalty from stalling the pipeline. Both issues arise because some guarding predicates may not be available when the instructions are ready to advance down the pipeline. One possible cause for the unresolved predicate is that, due to dynamic scheduling, a predicate-defining instruction may not have been executed yet. Another cause could be due to a potential long latency of the predicate-defining instructions. Most predicates are produced by compare instructions. Under normal implementation, compare instructions require a serialized propagation of bit-wise operations. Thus, as the clock frequency and the operand size increase, compare instructions could require multiple cycles to execute.

Detail Description Paragraph:
[0029] A first problem occurs during scheduling steps 158, 159 when a predicated instruction continuously waits in the reservation stations 130 for the predicate-defining instruction to finish. A second problem arises at the rename stage 155, 156 before the instructions enter the dynamic portion of the processor. With

multiple definitions assigned to a common register, which is guarded by different predicates, the renaming mechanism may need to stall when the predicates are not resolved. As a result, "bubbles" or stalls can be introduced in the pipeline.

Detail Description Paragraph:
[0030] For the baseline performance embodiment described above, when a predicate has not yet been produced, all instructions that depend on this predicate must wait in the reservation stations 130. Even if all the other source operands are available, the instruction cannot be executed until the predicate is ready. In situations where some predicates have not been resolved, the reservation stations 130 will start to pile up with those instructions having unresolved guarding predicates. As a result, the reservation stations 130 can become saturated quickly and induce backpressure on the pipeline. In other words, because of the unresolved predicates, the pipeline may stall due to the saturation of reservation station 130 entries, thereby causing performance losses.

Detail Description Paragraph:
[0031] On the compiler side, through compiler analysis, a variable is deemed live at a point of the control flow graph if the variable's value at that point can reach a subsequent use. The same variable can be defined elsewhere along another control flow path. These paths of multiple variable definitions can meet, resulting in overlapping variable lifetimes. When the compiler picks these paths for an if-converted region, the variable definitions are assigned to a common register, with the corresponding overlapping lifetimes guarded by different predicates. As this straight-line if-converted region is executed, the processor encounters several instructions which, guarded by different predicate registers, write to the same register. The left side 180 of FIG. 1C shows a variable with overlapping lifetimes in two definition paths 182,183. The variable is assigned to register r40, and after if-conversion 188, the variable is guarded by two different predicates p9, p3.

Detail Description Paragraph:
[0032] The performance of a dynamic execution processor can degrade with the above described predicated code sequence. When a consumer instruction reaches the rename stage 155, 156, the renaming of the common register becomes ambiguous if the guarding predicates of the defining instructions are not resolved. In the middle 190 of FIG. 1C, two add instructions, guarded by p9 and p3, assign their respective results to the same architectural register r40. After renaming 194, the result register is renamed to rB and rC, respectively. A mov instruction that uses or consumes the result register follows immediately in the pipeline. If the mov instruction enters the rename stage before predicates p9 and p3 are evaluated, then the processor cannot correctly determine whether to rename r40 to physical registers rB or rC. Therefore, the processor stalls the consumer instruction, the mov instruction before entering the mov instruction into the rename stage.

Detail Description Paragraph:
[0033] FIG. 2 illustrates where the instructions may have traveled in the pipeline 200. In FIG. 2, the add instructions have already advanced down the pipeline. As mentioned before, if predicates p9 and p3 have not yet been resolved, the mov instruction must wait indefinitely before the entering rename stage 210. After the predicates p9 and p3 become resolved, the mov instruction can then advance down the pipeline 200 into the rename stage 210 to rename the mov instruction source operand to rB or rC.

Detail Description Paragraph:
[0034] A consumer instruction is not required to wait for the resolution of all guarding predicates of the defining instructions as shown in FIG. 2A. The consumer instruction must only wait for the latest defining instruction that is guarded true. Therefore, the consumer instruction first waits for the predicate of the last of the defining instructions to become available 256. If the predicate of the last

of the defining instructions turns out true 258, the consumer instruction can immediately advance in the pipeline 200 and, in this example, use the physical register of the last defining instruction, despite the outcome of other defining instructions. If the last defining instruction is not true i.e. nullified, then the consumer instruction must wait for the predicate of the second-to-last defining instruction 260. The process repeats until a latest defining instruction is guarded true. This prioritized checking scheme for the predicate values affects performance depending on the order those values become available. It will be further appreciated that the instructions represented by the blocks in FIG. 2A is not required to be performed in the order illustrated, and that all the processing represented by the blocks may not be necessary to practice the invention.

Detail Description Paragraph:
[0035] According to baseline performance embodiment described above, the simple dynamic processor that runs predicated code could suffer from excessive pipeline stalls due to scheduling and renaming issues as described above. One alternative embodiment postpones the predicated instructions down the pipeline and resolves the predicated instructions without significant change to the existing dynamic execution microarchitecture.

Detail Description Paragraph:
[0037] One embodiment of the select-.mu.op mechanism includes register renaming in a processor model similar to subscripting a variable in a compiler. As described above, when a common defined register guarded by different predicates is renamed to different physical registers, a consumer instruction cannot rename the corresponding source register correctly until the predicates are resolved. The processor then dynamically introduces special operators named select-.mu.ops to defer the exact renaming resolution of physical registers. By injecting a select-.mu.op into the instruction stream, the select-.mu.op indicates that multiple renamed registers defined under different predicates may have reached a common use. The multiple renamed registers and the corresponding guarding predicates are assigned to the source operands of the select-.mu.op. A new renamed register allocated for the result of select-.mu.op can then be referenced by all subsequent consumer instructions. Upon execution of the select-.mu.op, the data from one of the renamed registers is assigned to the result accordingly.

Detail Description Paragraph:
[0038] With the select-.mu.op mechanism, the consumer instructions do not need to stall for the resolution of the guarding predicates of the defining instructions. At the rename stage, the consumer instructions can safely reference to the destination of the select-.mu.op, knowing that the select-.mu.op will, upon execution, choose the correct value among all the renamed registers. Thus, the renaming ambiguity is delayed and later gracefully deciphered via the execution the select-.mu.ops. In essence, using select-.mu.op postpones the resolution of the renaming ambiguity to the latter stages of the pipeline, hence allowing the renaming activity in the early stages to continue.

Detail Description Paragraph:
[0039] Two embodiments are shown in FIG. 4 and FIG. 5. The first embodiment, FIG. 4, has two predicated instructions assigned to r40 410 which are renamed to rB rC as the source operands 450. The exact syntax of the select-.mu.op is explained in more detail below. The second embodiment shown in FIG. 5 also has two predicated instructions 510, but the predicated instructions assign the result to two different registers r43 and r9. Both registers r43 and r9 have been assigned in a preceding cycle. Thus, two distinct select-.mu.ops are produced 550.

Detail Description Paragraph:
[0043] For an embodiment having four source operands, the processor can encounter two, three, or four instructions that define register R before generating a select-.mu.op to resolve renaming ambiguity for register R. The generation of select-

.mu.op is triggered by two conditions. First, each one of the defining instructions, except the first defining instruction, must be guarded by unresolved predicates. And second, because the first instruction defines the default identifier, the first instruction must be either: An un-predicated instruction, or a predicated instruction whose predicate has been resolved true, or a previously generated select-.mu.op.

Detail Description Paragraph:
[0044] Register R is renamed to different physical registers as R's defining instructions enter the rename stage. The physical identifiers are recorded by the renaming mechanism. When the select-.mu.op is to be generated, the recorded identifiers are copied to the source operands of the select-.mu.op. The s0 operand is copied with the physical identifier defined by the first instruction. The rest of one, two, or three physical identifiers fill the source operands in the order from s1 to s3. The processor then allocates a new physical register and assigns it to the destination (dest) operand. Thus, this format handles at most three parallel predicated instructions writing to the same register. Therefore, any of the four source operands is a candidate that potentially holds the final value, and the destination operand is where the final value is assigned. Once the select-.mu.op is formed, the processor inserts the select-.mu.op with the in-flight instructions and loads the select-.mu.op into the reservation station. The renaming unit, which does not need to wait for the resolution of the select-.mu.op, can then rename the subsequent uses of register R to the destination register of the select-.mu.op. The priority information of the source operands is inherent in the select-.mu.op, with s3 representing the highest priority. When the status bits of s3 indicate the operand is valid and ready, the select-.mu.op can immediately be executed without waiting on the resolution of the rest of the source operands. For one embodiment, the priority of the source operands is laid out, from left to right, in the program order that the instructions are fetched. Thus, the youngest defining instruction always has the highest priority.

Detail Description Paragraph:
[0045] One embodiment is a method 750 of processing predicated instructions as shown in FIG. 7A. First, receiving a plurality of predicated instructions assigned to a common defined register in block 752. At least one of the predicated instructions is out of order in a dynamic pipeline. Next, in block 754, the destination register for each one of the predicated instructions is renamed. Then, the renamed destination register with the predicate register of the predicated instruction is assigned to the source operand of a select-.mu.op, as shown in block 756. Next, a valid predicate is determined in block 758. The register corresponding to the select-.mu.op that corresponds to the valid predicate is selected in block 760. A consumer instruction is executed in block 762 wherein the consumer instruction uses the data from the register corresponding to the valid predicate. It will be further appreciated that the instructions represented by the blocks in FIG. 7A is not required to be performed in the order illustrated, and that all the processing represented by the blocks may not be necessary to practice the invention.

Detail Description Paragraph:
[0047] For one embodiment, the select-.mu.ops include use of a register alias table (RAT) with predicates. There are several approaches to support the generations of select-.mu.ops as described above. For one embodiment, the RAT is augmented and used in the rename stage with predicates. The RAT is used by the renaming unit to map from architectural register identifiers to physical register identifiers. When an in-flight instruction enters rename, the RAT looks up the physical identifiers of the source operands as well as assigns the result operand with a new physical identifier.

Detail Description Paragraph:
[0048] For one embodiment of the augmented RAT, each entry is expanded to have

multiple slots, with each slot recording the identifiers of the physical register as well as the guarding predicate of the instruction that defines this physical register. A logic view of the augmented RAT is shown in FIG. 8. Each row (entry) is assigned an architectural register whose identifier is used to index to the entry. Thus the number of architectural registers determines the number of rows in the RAT. For an embodiment of the RAT to support the select-.mu.ops with four source operands, each row of this table consists of a valid bit and four slots. Alternative embodiments with more or less source operands can similarly be constructed and used.

Detail Description Paragraph:
[0049] In the rename stage, the augmented RAT operates in three steps for the result register of an in-flight instruction. First, index into the RAT with the architectural identifier of the result register. Next, for the located entry, check the predicate of the instruction, i.e.: If the instruction is not predicated, clear the entire entry. If the predicate matches one of the predicates in the slots, clear its associated slot. Then, allocate a new physical register and append to a slot the physical identifier along with the identifier of the guarding predicate. A select-.mu.op is required only when two or more slots are occupied.

Detail Description Paragraph:
[0055] One of the guarding predicates in the slots is re-defined.

Detail Description Paragraph:
[0056] When any one of the above conditions is met, a select-.mu.op is generated. Physical identifiers in all of the occupied slots are copied to the source operands of the select-.mu.op. A new physical register is allocated for the destination operand. Then, the select-.mu.op is treated as an un-predicated instruction. That is, the entire entry in the RAT is cleared and replaced with the new physical register identifier.

Detail Description Paragraph:
[0057] For one embodiment, once a select-.mu.op is loaded into the reservation station like any other instruction, the reservation station holds the instructions and receives broadcasted data through the bypass network. When the select-.mu.op's source operands become available, the instruction can be dispatched.

Detail Description Paragraph:
[0058] For one embodiment of a dynamic execution model, the reservation station receives two bits of bypassed information for the status bits of the source operands in a select-.mu.op. One bit (bit1) signals that the computation of the operand has completed and the bypassed data is ready. Bit1 corresponds to the v-bit of the source operand. The other bit (bit2) indicates whether the bypassed data is to be committed or discarded, which is equivalent to the predicate of the result-producing instruction. Bit2 corresponds to the p-bit of the source operand. The status bits, v-bit and p-bit, in the select-.mu.op determine the select-.mu.op dispatch policy. One embodiment of the logic 900 that realizes the dispatching condition with the source fan-in of 4 is shown in FIG. 9. When the highest priority operand (s3) is available, v3 becomes 1. Depending on p3, which is the predicate value, the select-.mu.op can be immediately dispatched if p3 is 1. If p3 is 0, the select-.mu.op must wait for the select-.mu.op's lower priority operands to become available.

Detail Description Paragraph:
[0059] Once dispatched, the select-.mu.op is executed. The value from one of the source operands is transferred to the destination register. One embodiment of the logic 1000 that executes the select-.mu.op with source fan-in of 4 is shown in FIG. 10. This logic includes a cascade of three 2.times.1 multiplexers 1010, 1020, 1030. The p-bit is used to toggle the multiplexer select. Note that this is a logical view of the select-.mu.op execution. The actual circuitry can be implemented in

different ways, and an efficient implementation is needed to handle larger or smaller fan-ins. When a p-bit is set to 1, the output obtains the data from the corresponding source operand. Conversely when a p-bit is set to 0, the data is fetched from the output of another cascaded multiplexer. This logic 1000 correctly realizes the priority specified in the select-.mu.op. Once the execution of select-.mu.op completes, one of the source operands is assigned to the destination operand. The reservation station then receives the destination operand broadcast for all its uses.

Detail Description Paragraph:
[0065] The compiler cannot schedule (p7) add r40=0,r0 to be executed simultaneously with the other two predicated instructions. The architectural definition of IA-64 prevents a register, namely r40, from being assigned a value more than once in a single cycle. Since the compiler cannot guarantee that p7 (condition 2) and the other predicates (condition 1) are mutually exclusive, the compiler cannot schedule all three instructions in a single cycle. However, in the dynamic execution embodiment, executing those three instructions simultaneously is possible due to register renaming.

Detail Description Paragraph:
[0067] Once the instructions pass the renaming stage, all registers are renamed and each definition of a register is uniquely assigned a physical register. The registers in the pipeline have all numerical (architectural) register identifiers renamed to alphabetical (physical) register identifiers. In the pipeline diagram shown in FIGS. 12A-G, note that register r40, guarded by three different predicates, have also been renamed to rS, rT and rU.

Detail Description Paragraph:
[0068] After all registers have been renamed, the predicated register alias table (RAT) detects the renaming of r40, and dynamically attaches select-.mu.ops with the instruction bundle. Once the select-.mu.ops have been injected, the instructions enter the issue stage for dispersal. The issue unit disperses the instructions to several independent reservation stations. For one embodiment, the processor has a centralized reservation station dispatching instructions to two Integer functional units (I-unit) and two Memory functional units (M-unit). The reservation stations can dispatch any instruction when all except predicate dependencies are satisfied. The reason, as we previously mentioned, is that we can slip the predicated instructions and not commit their results until later when the predicate is known. We also assume that all integer operations take 1 cycle and load instructions 2 cycles. Since this paper does not deal with the dispersal rules of the issue unit, we simply assume a greedy algorithm that issues up to 4 instructions per cycles. FIGS. 12A-G illustrate benefits of select-.mu.op dynamic execution on the right side 1205 of each figure. Static execution is illustrated on the left side 1210 of each figure for comparison.

Detail Description Paragraph:
[0069] FIG. 12A shows cycle 0. In cycle 0, both rA and rB are the live-in registers, so after 1 cycle, an I-unit executes add rG=(. . .),rA and an M-unit executes ld2.acq rH=[rB]. Unlike static execution, since (pM) add rT=0,r0 does not depend on any register except the predicate; (pM) add rT=0,r0 also gets dispatched, but does not get committed until pM is known.

Detail Description Paragraph:
[0070] FIG. 12B shows cycle 1. After Cycle 1, rG becomes available and triggers the reservation station to dispatch ld2 rJ=[rG] to an M-unit. Since the load instructions take two cycles, ld2.acq rH=[rB] in the other M-unit will not be ready until after Cycle 2. Again, (pN) add rS=12,r0 is still not committed, and for the same reason as before, both I-units are to execute (pL) add rU=0,r0 and (pN) add rS=12,r0.

Detail Description Paragraph:
[0071] FIG. 12C shows cycle 2. After Cycle 2, rH is available, rC is a live-in. Thus and rK=rH,rC can be dispatched to an I-unit. The register rJ is still pending. One of the M-units will be free. The reorder buffer does not retire (pM) add rT=0,r0 because the predicate pM has not been evaluated.

Detail Description Paragraph:
[0072] FIG. 12D shows cycle 3. After Cycle 3, both rK and rJ are ready. Thus, both of the compare instructions can be dispatched. Also, all three predicated instructions now wait in the reorder buffer for the predicates to be resolved.

Detail Description Paragraph:
[0073] FIG. 12E shows cycle 4. Several actions take place after Cycle 4. First, all three predicates pM, pN, and pL have been calculated. The predicate dependencies are resolved and all three predicated instructions can immediately be committed.

CLAIMS:

1. A microprocessor comprising: a plurality of dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes a plurality of guarding predicates; a register renaming unit; a reorder buffer; a plurality of execution units; a plurality of reservation stations wherein the register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the plurality of dynamic pipeline stages; and an augmented register alias table.

9. A method of processing predicated instructions comprising: receiving a plurality of predicated instructions assigned to a common defined destination register and wherein at least one of the plurality of predicated instructions is out of order in an dynamic pipeline; renaming the destination register for each one of the plurality of predicated instructions; assigning the corresponding renamed destination register for each one of the plurality of predicated instructions with a corresponding predicate register to corresponding ones of the a plurality of source operands of a select-.mu.op; determining a valid predicate in the source operands of the select-.mu.op; electing the register corresponding to the select-.mu.op that corresponds to the valid predicate; transferring the data in the selected register to the destination register; and executing a consumer instruction wherein the consumer instruction uses the data from the destination register of the corresponding select-.mu.op.

14. A computer system comprising: a processor, wherein the processor includes: a plurality of dynamic pipeline stages including at least one predicated instruction wherein the predicated instruction includes a plurality of guarding predicates; a register renaming unit; a reorder buffer; a plurality of execution units; a plurality of reservation stations wherein the register renaming unit, the reorder buffer, the plurality of execution units and the plurality of reservation stations are coupled to at least one of the plurality of dynamic pipeline stages; and an augmented register alias table; a system bus; a computer memory system; an input/output device; wherein the system bus is coupled to the processor, the computer memory system and the input/output device.

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L12: Entry 1 of 1 File: USPT Jan 28, 2003

DOCUMENT-IDENTIFIER: US 6513109 B1
TITLE: Method and apparatus for implementing execution predicates in a computer processing system

Detailed Description Text (13):
The memory subsystem 305 stores the program instructions. The instruction cache 310 acts as a high-speed access buffer for the instructions. When instructions are not available in the instruction cache 310, the instruction cache 310 is filled using the contents of the memory subsystem 305. At each cycle, using the program counter 365, the instruction cache 310 is queried by instruction fetch logic (not shown) for the next set of instructions. The instruction fetch logic optionally includes branch prediction logic and branch prediction tables (not shown). In response to the query, the instruction cache 310 returns one or more instructions, which then enter the instruction buffer 325 in the same relative order. For each instruction in the instruction buffer 325, the Instruction Address (IA) is also held. The logic associated with the instruction buffer 325 allocates an entry in the retirement queue 355 (which is also referred to as a reorder buffer) for each of the instructions. The ordering of the entries in the retirement queue 355 is relatively the same as that of instruction fetch. The retirement queue 355 is described more fully hereinbelow.

Detailed Description Text (14):
The instruction buffer 325 copies the instructions to the dispatch unit 330, which performs the following multiple tasks: 1. The dispatch unit 330 decodes each instruction. 2. The dispatch unit 330 renames the destination operand specifier (s) in each instruction, in which the architected destination register operand specifiers are mapped to distinct physical register specifiers in the processor. The physical register specifiers specify the registers in the future register file 335 and architected register file 360. The dispatch unit 330 also updates relevant tables associated with the register rename logic (not shown). 3. The dispatch unit 330 queries the architected register file 360, and the future register file 335, in that order, for the availability of any source operands of the type general-register, floating-point register, condition register, or any other register type. If any of the operands are currently available in the architected register file 360, their values are copied along with the instruction. If the value of the operand is not currently available (because it is being computed by an earlier instruction), then the architected register file 360 stores an indication to that effect, upon which the future register file 335 is queried for this value using the physical register name (which is also available from the architected register file 360) where the value was to be available. If the value is found, then the value is copied along with the instruction. If the value is not found, then the physical register name where the value is to become available in the future is copied. 4. The dispatch unit 330 dispatches the instructions for execution to the following execution units: the branch unit 340; the functional units 345; and the memory units 350. The decision as to which instruction will enter a Reservation Station (not shown) (organized as a queue) for execution is made based on the execution resource requirements of the instruction and the availability of the execution units.

Detailed Description Text (15):
Once all the source operand values necessary for the execution of an instruction are available, the instruction executes in the execution units and the result value is computed. The computed values are written to the retirement queue 355 entry for the instruction (further described below), and any instructions waiting in the reservation stations for the values are marked as ready for execution.

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L12: Entry 1 of 1 File: USPT Jan 28, 2003

US-PAT-NO: 6513109
DOCUMENT-IDENTIFIER: US 6513109 B1

TITLE: Method and apparatus for implementing execution predicates in a computer processing system

DATE-ISSUED: January 28, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gschwind; Michael K.	Danbury	CT		
Sathaye; Sumedh	Fishkill	NY		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 09/ 387220 [PALM]
DATE FILED: August 31, 1999

INT-CL: [07] G06 F 9/00

US-CL-ISSUED: 712/200; 712/233
US-CL-CURRENT: 712/200; 712/233

FIELD-OF-SEARCH: 712/200, 712/220, 712/233, 712/23

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5632023	May 1997	White et al.	712/215
☐	5758051	May 1998	Moreno et al.	714/2
☐	5799179	August 1998	Ebcioglu et al.	712/234
☐	5901308	May 1999	Cohn et al.	712/216
☐	5999738	December 1999	Schlansker et al.	712/234
☐	6260189	July 2001	Batten et al.	717/151

☐	6442679	August 2002	Klauser et al.	712/218

OTHER PUBLICATIONS

Mahlke et al., "Sentinel Scheduling for VLIW and Superscalar Processors", Proceedings of the 5.sup.th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 1992.
Park et al., "On Predicated Execution", Tech. Rep. HPL-91-58, HP Laboratories, Palo Alto, CA, May 1991.
Rau, et al., "The Cydra 5 Departmental Supercomputer Design Philosophies, Decisions, and Trade-offs", IEEE Computer Society, vol. 22, No. 1, Jan. 1989.
David C. Lin, "Compiler Support For Predicated Execution In Superscalar Processors", Thesis, Master of Science in Electrical Engineering, Graduate College, University of Illinois, 1990.
Mahlke et al., "Effective Compiler Support for Predicated Execution Using the Hyperblock", IEEE, Sep. 1992, pp. 45-54.
K. Ebcioglu, "Some Design Ideas for a VLIW Architecture for Sequential-Natured Software, Parallel Processing", Cosnard et al. (Editors), Proceedings of the IFIP WG 10.3 Working Conference on Parallel Processing, Pisa Italy, Apr. 25-27, 1988, pp. 3-21.
Ebcioglu et al., "Some Global Compiler Optimizations and Architectural Features for Improving Performance of Superscalars", Research Report, RC 16145 (#71759) Oct. 2, 1990, pp. 1-13.
Ebcioglu et al., "An Eight-Issue Tree-VLIW Processor for Dynamic Binary Translation" IEEE Feb. 1998, pp. 488-495.
Smith et al., "Implementing Precise Interrupts in Pipelined Processors", IEEE Transactions on Computers, vol. 37, No. 5, May 1988, pp. 562-573.
Pnevmatikatos et al., "Guarded Execution and Branch Prediction in Dynamic ILP Processors", 21.sup.st International Symposium on Computer Architecture, Chicago, IL, 1994.

ART-UNIT: 2183

PRIMARY-EXAMINER: Coleman; Eric

ATTY-AGENT-FIRM: F. Chau & Associates, LLP

ABSTRACT:

There is provided a method for executing an ordered sequence of instructions in a computer processing system. The sequence of instructions is stored in a memory of the system. At least one of the instructions includes a predicated instruction that represents at least one operation that is to be conditionally performed based upon an associated flag value. The method includes the step of fetching a group of instructions from the memory. Execution of instructions are scheduled within the group, wherein the predicated instruction is moved from its original position in the ordered sequence of instructions to an out-of-order position in the sequence of instructions. The instructions are executed in response to the scheduling. In one embodiment of the invention, the method further includes generating a predicted value for the associated flag value, when the associated flag value is not available at execution of the predicated instruction. In another embodiment, the method further includes modifying execution of the operations represented by the predicated instruction based upon the predicted value. In yet another embodiment, the modifying step includes selectively suppressing either the execution or write back of results generated by the operations represented by the predicated instruction based upon the predicted value. In still another embodiment, the method includes predicting a data dependence relationship of an instruction with a previous predicated instruction or another previous instruction. The correctness of

the relationship prediction may be verified, and a selection may be made from among a number of predicted dependencies.

35 Claims, 11 Drawing figures

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L12: Entry 1 of 1 File: USPT Jan 28, 2003

US-PAT-NO: 6513109
DOCUMENT-IDENTIFIER: US 6513109 B1

TITLE: Method and apparatus for implementing execution predicates in a computer processing system

DATE-ISSUED: January 28, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Gschwind; Michael K.	Danbury	CT		
Sathaye; Sumedh	Fishkill	NY		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 09/ 387220 [PALM]
DATE FILED: August 31, 1999

INT-CL: [07] G06 F 9/00

US-CL-ISSUED: 712/200; 712/233
US-CL-CURRENT: 712/200; 712/233

FIELD-OF-SEARCH: 712/200, 712/220, 712/233, 712/23

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5632023	May 1997	White et al.	712/215
☐	5758051	May 1998	Moreno et al.	714/2
☐	5799179	August 1998	Ebcioglu et al.	712/234
☐	5901308	May 1999	Cohn et al.	712/216
☐	5999738	December 1999	Schlansker et al.	712/234
☐	6260189	July 2001	Batten et al.	717/151

☐	6442679	August 2002	Klauser et al.	712/218

OTHER PUBLICATIONS

Mahlke et al., "Sentinel Scheduling for VLIW and Superscalar Processors", Proceedings of the 5.sup.th International Conference on Architectural Support for Programming Languages and Operating Systems, Oct. 1992.
Park et al., "On Predicated Execution", Tech. Rep. HPL-91-58, HP Laboratories, Palo Alto, CA, May 1991.
Rau, et al., "The Cydra 5 Departmental Supercomputer Design Philosophies, Decisions, and Trade-offs", IEEE Computer Society, vol. 22, No. 1, Jan. 1989.
David C. Lin, "Compiler Support For Predicated Execution In Superscalar Processors", Thesis, Master of Science in Electrical Engineering, Graduate College, University of Illinois, 1990.
Mahlke et al., "Effective Compiler Support for Predicated Execution Using the Hyperblock", IEEE, Sep. 1992, pp. 45-54.
K. Ebcioglu, "Some Design Ideas for a VLIW Architecture for Sequential-Natured Software, Parallel Processing", Cosnard et al. (Editors), Proceedings of the IFIP WG 10.3 Working Conference on Parallel Processing, Pisa Italy, Apr. 25-27, 1988, pp. 3-21.
Ebcioglu et al., "Some Global Compiler Optimizations and Architectural Features for Improving Performance of Superscalars", Research Report, RC 16145 (#71759) Oct. 2, 1990, pp. 1-13.
Ebcioglu et al., "An Eight-Issue Tree-VLIW Processor for Dynamic Binary Translation" IEEE Feb. 1998, pp. 488-495.
Smith et al., "Implementing Precise Interrupts in Pipelined Processors", IEEE Transactions on Computers, vol. 37, No. 5, May 1988, pp. 562-573.
Pnevmatikatos et al., "Guarded Execution and Branch Prediction in Dynamic ILP Processors", 21.sup.st International Symposium on Computer Architecture, Chicago, IL, 1994.

ART-UNIT: 2183

PRIMARY-EXAMINER: Coleman; Eric

ATTY-AGENT-FIRM: F. Chau & Associates, LLP

ABSTRACT:

There is provided a method for executing an ordered sequence of instructions in a computer processing system. The sequence of instructions is stored in a memory of the system. At least one of the instructions includes a predicated instruction that represents at least one operation that is to be conditionally performed based upon an associated flag value. The method includes the step of fetching a group of instructions from the memory. Execution of instructions are scheduled within the group, wherein the predicated instruction is moved from its original position in the ordered sequence of instructions to an out-of-order position in the sequence of instructions. The instructions are executed in response to the scheduling. In one embodiment of the invention, the method further includes generating a predicted value for the associated flag value, when the associated flag value is not available at execution of the predicated instruction. In another embodiment, the method further includes modifying execution of the operations represented by the predicated instruction based upon the predicted value. In yet another embodiment, the modifying step includes selectively suppressing either the execution or write back of results generated by the operations represented by the predicated instruction based upon the predicted value. In still another embodiment, the method includes predicting a data dependence relationship of an instruction with a previous predicated instruction or another previous instruction. The correctness of

the relationship prediction may be verified, and a selection may be made from among a number of predicted dependencies.

35 Claims, 11 Drawing figures

First Hit Fwd Refs

Generate Collection Print

L10: Entry 11 of 18 File: USPT Dec 28, 1999

US-PAT-NO: 6009512
DOCUMENT-IDENTIFIER: US 6009512 A

TITLE: Mechanism for forwarding operands based on predicated instructions

DATE-ISSUED: December 28, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Christie; David S.	Austin	TX		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 08/ 958477 [PALM]
DATE FILED: October 27, 1997

INT-CL: [06] G06 F 9/318

US-CL-ISSUED: 712/226; 712/218
US-CL-CURRENT: 712/226; 712/218

FIELD-OF-SEARCH: 395/582, 395/581, 395/583, 395/567, 395/394, 712/235, 712/234, 712/236, 712/226, 712/218

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5471593	November 1995	Branigin	712/235

OTHER PUBLICATIONS

Mahlke, Scott A. et al., A Comparison of Full and Partial Predicated Execution Support for ILP Processors, 1995, ACM, pp. 138-149.
Johnson, Richard et al., Analysis Techniques for Predicated Code, 1996, IEEE, pp. 100-113.
Tyson, Gary Scott, The Effects of Predicated Execution on Branch Prediction, 1994, ACM, pp. 196-206.
August, David J. et al., A Framework for Balancing Control Flow and Predication, 1997, IEEE, pp. 92-103.

Kathail et al., HPL Playdoh Architecture Specification: Version 1.0, Hewlett Packard, Computer Systems Laboratory, HPL-93-80, Feb., 1994, pp. 1-48.

ART-UNIT: 273

PRIMARY-EXAMINER: Ellis; Richard L.

ATTY-AGENT-FIRM: Conley, Rose & Tayon Merkel; Lawrence J. Kivlin; B. Noel

ABSTRACT:

A method and apparatus for providing predicated instructions in a processor employing out of order execution. In one embodiment, a plurality of decode units are configured to decode a plurality of variable byte length instructions and to provide a plurality of output of signals. The output signals are provided to a plurality of reservation stations coupled to the plurality of decode units within the superscalar microprocessor. Functional units are configured to receive the output signals from the plurality of decode units. The functional units include function execution units coupled to receive signals from the plurality of reservation stations and to provide a function output responsive to the output signals. The functional units further comprise a predication unit configured to determine whether a predetermined condition has occurred and either stop the function output or allow the function output to be transmitted depending on whether the predetermined condition has occurred.

22 Claims, 14 Drawing figures

First Hit Fwd Refs

Generate Collection Print

L10: Entry 11 of 18 File: USPT Dec 28, 1999

US-PAT-NO: 6009512
DOCUMENT-IDENTIFIER: US 6009512 A

TITLE: Mechanism for forwarding operands based on predicated instructions

DATE-ISSUED: December 28, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Christie; David S.	Austin	TX		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 08/ 958477 [PALM]
DATE FILED: October 27, 1997

INT-CL: [06] G06 F 9/318

US-CL-ISSUED: 712/226; 712/218
US-CL-CURRENT: 712/226; 712/218

FIELD-OF-SEARCH: 395/582, 395/581, 395/583, 395/567, 395/394, 712/235, 712/234, 712/236, 712/226, 712/218

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5471593	November 1995	Branigin	712/235

OTHER PUBLICATIONS

Mahlke, Scott A. et al., A Comparison of Full and Partial Predicated Execution Support for ILP Processors, 1995, ACM, pp. 138-149.
Johnson, Richard et al., Analysis Techniques for Predicated Code, 1996, IEEE, pp. 100-113.
Tyson, Gary Scott, The Effects of Predicated Execution on Branch Prediction, 1994, ACM, pp. 196-206.
August, David J. et al., A Framework for Balancing Control Flow and Predication, 1997, IEEE, pp. 92-103.

Kathail et al., HPL Playdoh Architecture Specification: Version 1.0, Hewlett Packard, Computer Systems Laboratory, HPL-93-80, Feb., 1994, pp. 1-48.

ART-UNIT: 273

PRIMARY-EXAMINER: Ellis; Richard L.

ATTY-AGENT-FIRM: Conley, Rose & Tayon Merkel; Lawrence J. Kivlin; B. Noel

ABSTRACT:

A method and apparatus for providing predicated instructions in a processor employing out of order execution. In one embodiment, a plurality of decode units are configured to decode a plurality of variable byte length instructions and to provide a plurality of output of signals. The output signals are provided to a plurality of reservation stations coupled to the plurality of decode units within the superscalar microprocessor. Functional units are configured to receive the output signals from the plurality of decode units. The functional units include function execution units coupled to receive signals from the plurality of reservation stations and to provide a function output responsive to the output signals. The functional units further comprise a predication unit configured to determine whether a predetermined condition has occurred and either stop the function output or allow the function output to be transmitted depending on whether the predetermined condition has occurred.

22 Claims, 14 Drawing figures

L10: Entry 11 of 18 File: USPT Dec 28, 1999

DOCUMENT-IDENTIFIER: US 6009512 A
TITLE: Mechanism for forwarding operands based on predicated instructions

Abstract Text (1):
A method and apparatus for providing predicated instructions in a processor employing out of order execution. In one embodiment, a plurality of decode units are configured to decode a plurality of variable byte length instructions and to provide a plurality of output of signals. The output signals are provided to a plurality of reservation stations coupled to the plurality of decode units within the superscalar microprocessor. Functional units are configured to receive the output signals from the plurality of decode units. The functional units include function execution units coupled to receive signals from the plurality of reservation stations and to provide a function output responsive to the output signals. The functional units further comprise a predication unit configured to determine whether a predetermined condition has occurred and either stop the function output or allow the function output to be transmitted depending on whether the predetermined condition has occurred.

Brief Summary Text (2):
This invention relates to implementing conditional instructions in a computer system and, more particularly, to a method and apparatus for providing support for predicated instructions in an X86 architecture.

Brief Summary Text (14):
Another method for selectively performing operations is to use predicated instructions. Predicated instructions build the condition-checking task into individual instructions such that when they are executed, the operation is performed only if the specified condition is met. If the condition is not met, the operation is canceled, effectively making the instruction a no-op. The primary difference between conditional jumps and predicated instructions is that a conditional jump can instantly cancel a large number of instructions (including instructions which will be refetched and reexecuted). However, predicated instructions self-cancel on a one-by-one basis. In a pipelined processor, conditional jumps cause wasted cycles even with branch prediction if the conditional jump is mispredicted and the target address lies within the sequential path already fetched.

Brief Summary Text (15):
The use of conditional jumps versus predicated instructions is illustrated in Examples 1 and 2. Example 1 illustrates the use of a conditional jump to selectively perform certain operations (applying floor and ceiling functions to the elements of an array and maintaining counts of the number clipped).

Brief Summary Text (19):
The same operations using predicated instructions (denoted by the suffixes .GT and .LE) is illustrated in Example 2 below:

Brief Summary Text (21):
Predicated Instruction

Brief Summary Text (22):
Predicated operations are always fetched and sent to execution, but are only carried out if the status flags set by the preceding compare instruction are in the proper state. Otherwise, they are effectively no-ops. In a scalar processor, each no-op is an idle cycle. For each compare that would have resulted in a taken branch, there are two idle cycles. But, since a cycle is eliminated by removing the branch instructions, the net loss is effectively one cycle, which is less than the pipe refill penalty.

Brief Summary Text (23):
In a superscalar implementation, the two predicated operations after each compare can be executed in parallel, not only with each other, but with other instructions in the sequence with no wasted cycles at all. Moreover, the two clipping operations can always be done in parallel, given sufficient execution resources, and will only be done once per element. The lower clipping operation need never be canceled and redone because of a mispredict on the upper one.

Brief Summary Text (24):
As processor speeds increase (and pipeline lengths increase as well), it is increasingly desirable to avoid the pipeline penalty imposed by conditional jumps. At the same time, however, it is desired to limit software overhead. Accordingly, there is a need for an X86 compatible processor employing predicated instructions with a minimum of software overhead.

Brief Summary Text (26):
Accordingly, a method and apparatus for providing predicated instructions in a processor employing out of order execution is provided herein. In one embodiment, a plurality of decode units are configured to decode a plurality of variable byte length instructions and to provide a plurality of output of signals. The output signals are provided to a plurality of reservation stations coupled to the plurality of decode units within the superscalar microprocessor. Functional units are configured to receive the output signals from the plurality of decode units. The functional units include function execution units coupled to receive signals from the plurality of reservation stations and to provide a function output responsive to the output signals. The functional units further comprise a predication unit configured to determine whether a predetermined condition has occurred and either stop the function output or allow the function output to be transmitted depending on whether the predetermined condition has occurred.

Brief Summary Text (27):
In one specific embodiment, an instruction encoding mechanism such as prefix bytes are provided for specifying a condition code upon which the instruction is predicated. Status flags in an EFLAGS register of the microprocessor are set by a particular instruction with subsequent instructions executed conditionally as controlled by the predication unit conditional execution encoding.

Brief Summary Text (28):
In another specific embodiment, predicated instruction execution is accomplished by providing a set of single bit predicate flags to control conditional execution and a mechanism for setting the flags based on status flag results. Thus the true/false value of a condition code is transferred to a predicate flag for storage to be used at a later time. The EFLAGS register may be used to accommodate the predicate flags. The predicate flags are set by specifying a condition code and a predicate flag to record the condition. For example, the SETcc instruction may be used to set or clear one of the predicate flags. An instruction which sets the status flags in the EFLAGS register is followed by one or more SETcc instructions to transfer the desired condition codes states to predicate flags. Alternatively, instruction prefixes may be defined that specify a condition code and predicate flag to be written based on the result of the associated instruction. Predication of an instruction is controlled by a prefix that specifies a particular predicate flag.

Brief Summary Text (29):
Finally, a third specific embodiment employs a combination of condition codes and predicate flags. Two sets of predicate prefixes are employed: one to specify use of a predicate flag and one to specify a condition code.

Drawing Description Text (6):
FIG. 4 is a flowchart illustrating operation of a typical predicated instruction type process;

Drawing Description Text (8):
FIG. 6 is a diagram illustrating a pipeline for a predicated instruction in a processor employing out of order execution;

Drawing Description Text (9):
FIG. 7 illustrates a condition code based predicated instruction;

Drawing Description Text (10):
FIG. 8 illustrates an EFLAGS register with additional status flags for use with a condition code based predicated instruction;

Drawing Description Text (11):
FIG. 9 illustrates a condition code based predicated instruction with extended status flags specifies for use with the EFLAGS register of FIG. 7;

Drawing Description Text (12):
FIG. 10 illustrates an EFLAGS register including predicate flags;

Drawing Description Text (13):
FIG. 11 illustrates one method of setting the predicated flags of the EFLAGS register of FIG. 10 according to one embodiment of the present invention;

Drawing Description Text (14):
FIG. 12 illustrates a set predicate flag instruction to permit the setting of the predicate flags in the EFLAGS register of FIG. 10 according to one embodiment of the present invention;

Drawing Description Text (15):
FIG. 13 illustrates a predicated instruction identifying a predicate flag or flags according to one embodiment of the present invention;

Drawing Description Text (16):
FIG. 14 illustrates a predicated instruction including both condition code and predicate flags according to one embodiment of the present invention.

Detailed Description Text (3):
FIG. 4--Flowchart of Predicated Instruction Execution

Detailed Description Text (4):
FIG. 4 illustrates the operation of the predicated instruction mechanism. Initially, an operation is performed which updates the EFLAGS register (step 50). Subsequently, a predicated instruction is executed. The predicated instruction is read (step 52). If the status flags (or predicate flags) have been set in step 50 to the proper state, then the predicated instruction is executed (step 56). If, however, the status flags were not in the proper state, then an idle or no-op operation occurs (step 58).

Detailed Description Text (5):
FIG. 5--Microprocessor Employing Predicated Instruction Mechanism

Detailed Description Text (6):
Referring next to FIG. 5, a block diagram of a microprocessor 1200 including a predicated instruction mechanism is shown. As illustrated in FIG. 5, superscalar microprocessor 1200 (preferably an x86-type processor) includes a prefetch/predecode unit 1202 and a branch prediction unit 1120 coupled to an instruction cache 1204. A main memory, typically separate from processor 1200, is coupled to processor 1200 by way of prefetch/predecode unit 1202. Instruction alignment unit 1206 is coupled between instruction cache 1204 and a plurality of decode units 1208A-1208C (referred to collectively as decode units 1208). Each decode unit 1208A-1208C is coupled to a respective reservation station units 1210A-1210C (referred to collectively as reservation stations 1210), and each reservation station 1210A-1210C is coupled to a respective functional unit 1212A-1212C (referred to collectively as functional units 1212). Decode units 1208, reservation stations 1210, and functional units 1212 are further coupled to a reorder buffer 1216, a register file 1218 and a load/store unit 1222. A data cache 1224 is finally shown coupled to load/store unit 1222, and an MROM unit 1209 is shown coupled to instruction alignment unit 1206 and decode units 1208. Each functional unit 1212 includes a predication unit 1213A-1213C (referred to collectively as predication unit 1213), whose use will be described in greater detail below.

Detailed Description Text (12):
Before proceeding with a detailed description of the predicated instruction mechanism, general aspects regarding other subsystems employed within the exemplary superscalar microprocessor 1200 of FIG. 5 will be described. For the embodiment of FIG. 5, each of the decode units 1208 includes decoding circuitry for decoding the predetermined fast path instructions referred to above. In addition, each decode unit 1208A-1208F routes displacement and immediate data to a corresponding reservation station unit 1210A-1210F. Output signals from the decode units 1208 include bit-encoded execution instructions for the functional units 1212 as well as operand address information, immediate data and/or displacement data.

Detailed Description Text (13):
The superscalar microprocessor of FIG. 5 supports out of order execution, and thus employs reorder buffer 1216 to keep track of the original program sequence for register read and write operations, to implement register renaming, to allow for speculative instruction execution and branch misprediction recovery, and to facilitate precise exceptions. As will be appreciated by those of skill in the art, a temporary storage location within reorder buffer 1216 is reserved upon decode of an instruction that involves the update of a register to thereby store speculative register states. Reorder buffer 1216 may be implemented in a first-in-first-out configuration wherein speculative results move to the "bottom" of the buffer as they are validated and written to the register file, thus making room for new entries at the "top" of the buffer. Other specific configurations of reorder buffer 1216 are also possible, as will be described further below. If a branch prediction is incorrect, the results of speculatively-executed instructions along the mispredicted path can be invalidated in the buffer before they are written to register file 1218.

Detailed Description Text (14):
The bit-encoded execution instructions and immediate data provided at the outputs of decode units 1208A-1208F are routed directly to respective reservation station units 1210A-1210F. In one embodiment, each reservation station unit 1210A-1210F is capable of holding instruction information (i.e., bit encoded execution bits as well as operand values, operand tags and/or immediate data) for up to three pending instructions awaiting issue to the corresponding functional unit. It is noted that for the embodiment of FIG. 5, each decode unit 1208A-1208F is associated with a dedicated reservation station unit 1210A-1210F, and that each reservation station unit 1210A-1210F is similarly associated with a dedicated functional unit 1212A-1212F. Accordingly, three dedicated "issue positions" are formed by decode units 1208, reservation station units 1210 and functional units 1212. Instructions

aligned and dispatched to issue position 0 through decode unit 1208A are passed to reservation station unit 1210A and subsequently to functional unit 1212A for execution. Similarly, instructions aligned and dispatched to decode unit 1208B are passed to reservation station unit 1210B and into functional unit 1212B, and so on.

Detailed Description Text (15):
Upon decode of a particular instruction, if a required operand is a register location, register address information is routed to reorder buffer 1216 and register file 1218 simultaneously. Those of skill in the art will appreciate that the X86 register file includes eight 32 bit real registers (i.e., typically referred to as EAX, EBX, ECX, EDX, EBP, ESI, EDI and ESP). Reorder buffer 216 contains temporary storage locations for results which change the contents of these registers to thereby allow out of order execution. A temporary storage location of reorder buffer 1216 is reserved for each instruction which, upon decode, modifies the contents of one of the real registers. Therefore, at various points during execution of a particular program, reorder buffer 1216 may have one or more locations which contain the speculatively executed contents of a given register. If following decode of a given instruction it is determined that reorder buffer 1216 has previous location(s) assigned to a register used as an operand in the given instruction, the reorder buffer 1216 forwards to the corresponding reservation station either: 1) the value in the most recently assigned location, or 2) a tag for the most recently assigned location if the value has not yet been produced by the functional unit that will eventually execute the previous instruction. If the reorder buffer has a location reserved for a given register, the operand value (or tag) is provided from reorder buffer 1216 rather than from register file 1218. If there is no location reserved for a required register in reorder buffer 1216, the value is taken directly from register file 1218. If the operand corresponds to a memory location, the operand value is provided to the reservation station unit through load/store unit 1222.

Detailed Description Text (16):
Reservation station units 1210A-1210F are provided to temporarily store instruction information to be speculatively executed by the corresponding functional units 1212A-1212F. As stated previously, each reservation station unit 1210A-1210F may store instruction information for up to three pending instructions. Each of the six reservation stations 1210A-1210F contain locations to store bit-encoded execution instructions to be speculatively executed by the corresponding functional unit and the values of operands. If a particular operand is not available, a tag for that operand is provided from reorder buffer 1216 and is stored within the corresponding reservation station until the result has been generated (i.e., by completion of the execution of a previous instruction). It is noted that when an instruction is executed by one of the functional units 1212A-1212F, the result of that instruction is passed directly to any reservation station units 1210A-1210F that are waiting for that result at the same time the result is passed to update reorder buffer 1216 (this technique is commonly referred to as "result forwarding"). Instructions are issued to functional units for execution after the values of any required operand (s) are made available. That is, if an operand associated with a pending instruction within one of the reservation station units 1210A-1210F has been tagged with a location of a previous result value within reorder buffer 1216 which corresponds to an instruction which modifies the required operand, the instruction is not issued to the corresponding functional unit 1212 until the operand result for the previous instruction has been obtained. Accordingly, the order in which instructions are executed may not be the same as the order of the original program instruction sequence. Reorder buffer 1216 ensures that data coherency is maintained in situations where read-after-write dependencies occur.

Detailed Description Text (18):
Each of the functional units 1212 also provides information regarding the execution of conditional instructions to the branch prediction unit 1220. If a branch

prediction was incorrect, instruction cache 1204 flushes instructions not needed, and causes prefetch/predecode unit 1202 to fetch the required instructions from main memory. It is noted that in such situations, results of instructions in the original program sequence which occur after the mispredicted branch instruction are discarded, including those which were speculatively executed and temporarily stored in load/store unit 1222 and reorder buffer 1216. Exemplary configurations of suitable branch prediction mechanisms are well known.

Detailed Description Text (19):
Results produced by functional units 1212 are sent to the reorder buffer 1216 if a register value is being updated, and to the load/store unit 1222 if the contents of a memory location is changed. If the result is to be stored in a register, the reorder buffer 216 stores the result in the location reserved for the value of the register when the instruction was decoded. As stated previously, results are also broadcast to reservation station units 1210A-1210F where pending instructions may be waiting for the results of previous instruction executions to obtain the required operand values.

Detailed Description Text (25):
Turning now to FIG. 6, there is shown a predication mechanism for use in a processor such as processor 1200 implementing predicate execution and out of order instruction execution. Processor 1200 is, for example, an X86 compatible processor in which outstanding dependencies are resolved, as discussed above, by forwarding results into reservation station 102. Reservation station 102 may, for example, be reservation station 1210A, 1210B, or 1210C as shown in FIG. 5. An instruction that references a register that is the target of a predicated instruction has one of two possible dependencies: Either on the result of the predicated instruction if its condition for execution is met, or on the result of whichever prior instruction to the predicated instruction last updated the register. Thus, the predicated instructions are forwarded the original contents of the destination register as an input operand. FIG. 6 illustrates reservation station 102, which includes operand A 108, operand B 110, and predicated instruction 106. Additionally, FIG. 6 illustrates a functional unit 120, which includes predication unit 117 and execution unit 114. Functional unit 120 may, for example, be functional unit 1212A, 1212B, or 1212C. At the same time that the operation is performed the condition code or predicate flag is evaluated in evaluation unit 116, as will be discussed in more detail below. The result of the evaluation is used to control a multiplexer 118, which receives the result of execution unit 114 and the output from operand A 108 which stores the original contents of the destination operand. Multiplexer 118 selects either the result of the operation, if the predicate condition is true, or the input operand that is the original value of the destination, if the predicate condition is false. Multiplexer 118 is also coupled to functional unit 120. Further dependencies are handled by making subsequent instructions unconditionally dependent on the result of the predicated instruction without concern for the condition code or predicate flag.

Detailed Description Text (26):
If the predicate for a particular instruction is false, only the destination input operand 108 need be available and the instruction can be issued without waiting for any dependency on the other input operand to be resolved. Moreover, if a predicate for a particular instruction is known to be false when the instruction is decoded, dispatch logic 112 can avoid sending the instruction on to the functional unit 120 altogether simply by marking its re-order buffer entry as canceled Subsequent instructions would not detect dependencies on these canceled operations. It is noted that in an alternate embodiment, depending on how the instruction pointer is maintained, it is possible to avoid even using a reorder buffer entry.

Detailed Description Text (27):
FIG. 7--Condition Code Based Predicated Instruction

Detailed Description Text (28):
As noted above, predication information may be provided through either condition codes or predicate flags. Turning now to FIG. 7, there is shown a diagram of a condition code-based predicated instruction. The condition code-based predicated instruction includes a prefix 202, a condition code 204, an opcode 206, modR/M field 208, SIB field 210, displacement field 212 and immediate field 214. As with the generic X86 instruction, the prefix field, the modR/M field, SIB field, displacement field and the immediate field are optional. It is noted, however, that the condition code byte or bytes 204 may either be implemented as part of the opcode 206 or as part of prefix 202. Condition code 204 specifies the condition upon which the instruction is predicated. Status flags in the EFLAGS register are set by a particular instruction with subsequent instructions executed conditionally as controlled by the conditional execution encoding mechanism of the conditional code byte or bytes 204. Different instructions can specify different conditions for a given setting of the status flags in the EFLAGS register. For example:

Detailed Description Text (31):
Turning now to FIG. 8, there is shown a diagram of an EFLAGS register employing additional status flags. More particularly, two sets of additional status flags 216 and 218 are shown. Additional status flags 216 and 218 occupy the unused upper bytes of the EFLAGS register. In the example shown, the parity flag is omitted since it is very infrequently used so as to accommodate two full sets of status flags. The EFLAGS register is used for reasons of compatibility, i.e., so that the status flags are preserved on an interrupt by the existing X86 interrupt mechanism. It is noted that while two sets of additional status flags are shown in FIG. 8, preferably only one set is employed so as to permit, for example, expanded uses of other bits. Instructions which write the EFLAGS register specify in their opcodes, for example, which set of status flags to use. Alternatively, the condition code predicate prefix 204 in FIG. 7 may be expanded to identify which set of status flags is referenced.

Detailed Description Text (32):
As noted above, a predicated instruction is designated by a prefix which specifies the predicate. To be able to use the sixteen standard X86 conditions, sixteen different prefixes may be defined. Five redundant groups of eight single byte opcodes (PUSH, POP, INK, DEC AND XCHG) exist. These opcodes specify a general register in the lower three bits of the opcode and hence take up eight opcode positions per instruction. However, each of these instructions has a two byte modR/M style equivalent. Accordingly, these opcodes can be redefined to expand architectural functionality without losing any existing functionality. The sixteen predicate prefixes may be made available using two sets of the redundant single byte opcodes described above

Detailed Description Text (33):
FIG. 9--Predicated Instruction with Extended Status Flag Specifiers

Detailed Description Text (34):
FIG. 9 illustrates the condition code based predicated instruction with extended status flags specifiers. The predicated instruction includes prefix byte or bytes 220, a condition code prefix 222, a status flag identifier 224, opcode 226, modR/M field 228, SIB field 230, displacement field 232, and immediate field 234. As above, the prefix 220, modR/M field 228, SIB field 230, displacement field 232, and immediate field 234 are optional. The condition code field 222 and status flag identifier field 224 may be part of the opcode 226 or part of the prefix 220.

Detailed Description Text (35):
FIG. 10--EFLAGS Register Employing Predicate Flags

Detailed Description Text (36):
An alternative embodiment uses a predicate flag based predicated execution. This

embodiment of predicated execution provides a set of single bit predicate flags to control conditional execution and an instruction encoding means for setting the EFLAGS based on status flag results. Essentially, the true/false value of a condition code is transferred to a predicate flag for storage to be used at a later time. Again, for backwards compatibility, the available space in the EFLAGS register is used and can accommodate eight predicate flags.

Detailed Description Text (37):
FIG. 11--Encoding Predicate Flags

Detailed Description Text (38):
One method of encoding the predicate flags is by redefining the SETCC instructions, which already specify the sixteen condition codes, to set or clear one of the eight predicate flags rather than writing one of the general registers. Turning now to FIG. 11, the process of setting one of the predicate flags employing the SETCC command is shown. Initially, an instruction is executed that will set the status flags (step 300). Next, the SETCC command will be executed (step 302). Initially, the status flags will be evaluated (step 304). If the appropriate condition has been met, the SETCC command will set the specified predicate flag to one (step 310). If, on the other hand, the relevant condition in step 306 was not met, then the predicate flag would be set to zero (step 308). In this fashion, the condition code is specified and a predicate flag records the condition. Thus, an instruction which sets the status flags is followed by one or more SETCC instructions to transfer the desired condition code states to predicate flags.

Detailed Description Text (39):
An alternate method for setting predicate flags is to define instruction prefixes that specify a particular condition code and a predicate flag to be written based on the result of the associated predicated instruction. The prefixes may be made available by redefining redundant single byte instruction encodings such as exist as discussed above for PUSH, POP, INC, and DEC.

Detailed Description Text (40):
FIG. 12--Set Predicate Flag Instruction

Detailed Description Text (41):
Turning now to FIG. 12, there is shown a diagram of a set predicate flag instruction. Set predicate flag instruction includes an optional prefix field 350, a condition code field 352, a predicate flag specifier field 354, and an opcode field 356. The condition code field 352 and the predicate field specifier 354 may be part of either opcode 356 or prefix 350. In addition, the set predicate flag instruction includes an optional modR/M 358, SIB field 360, displacement field 352, and immediate field 364. The condition code prefix 352 specifies one of the sixteen condition codes and the predicate flag prefix 354 specifies the destination predicate flag. This method allows the setting of a predicate flag to be accomplished in a single instruction rather than by a separate instruction. However, it is noted that the SETCC form may be used to report multiple conditions from one instruction. The use of predicate setting prefixes allows the setting of a predicate flag without updating the status flags in the EFLAGS register.

Detailed Description Text (42):
FIG. 13--Predicated Instruction Identifying Predicate Flags

Detailed Description Text (43):
Turning now to FIG. 13, there is shown a predicated instruction identifying the predicate flags. The predicated instruction includes an optional prefix field 366, a predicate flag field 368, an opcode field 370, an optional modR/M byte 372, an optional SIB byte 374, an optional displacement field 376, and an optional immediate field 380. The predicate flag prefix 368 specifies a particular flag. It is often desirable to have the complement of the condition available as well as the

condition itself. While this could be handled via two SETCC instructions on two predicate flags, a more efficient use of the EFLAGS may be to specify the complement of a predicate when it is used. To this end, one flag and a second set of eight prefixes may be defined. Thus, the set predicate flag prefixes illustrated in FIG. 11 need only specify the eight true conditions, not their complement. Thus, a total of four sets of eight prefixes to set one of eight flags for one of eight conditions and test the true or complement of each of the eight flags is defined. (Thus, the predicate flag field 368 in predicated instruction may include two bytes.) These prefixes could be made available using the redundant single byte instruction encodings such as exist for PUSH, POP, INC and the like.

Detailed Description Text (44):
FIG. 14--Predicated Instruction Employing Condition Codes and Predicate Flags

Detailed Description Text (45):
It is noted that in an alternative embodiment a combination of both methods is employed using the existing set of status flags plus eight predicate flags. Two sets of predicate prefixes are defined, one to specify the use of a predicate flag, and one to specify a condition based on the current state of the status flags. Such a predicated instruction is illustrated in FIG. 14. The predicated instruction including both condition code and predicate flag prefixes encodes a prefix field 382, a predicate flag field 384, condition code field 386, opcode 388, optional modR/M field 390, optional SIB field 392, optional displacement field 394, and optional immediate field 396. It is noted that predication could apply to branch operations as well as ALU and load store operations, including call, return, and jump indirect, all of which would have the same functionality as the JCC instruction.

Other Reference Publication (1):
Mahlke, Scott A. et al., A Comparison of Full and Partial Predicated Execution Support for ILP Processors, 1995, ACM, pp. 138-149.

Other Reference Publication (2):
Johnson, Richard et al., Analysis Techniques for Predicated Code, 1996, IEEE, pp. 100-113.

Other Reference Publication (3):
Tyson, Gary Scott, The Effects of Predicated Execution on Branch Prediction, 1994, ACM, pp. 196-206.

CLAIMS:

1. A method for performing predicated instruction execution, comprising:

encoding a first instruction with a condition code that identifies a condition specified by a status bit in a status register;

executing a second instruction, wherein said executing includes setting said status bit in said status register;

executing said first instruction, wherein said executing said first instruction includes:

decoding said condition code;

reading said status bit responsive to said decoding;

performing a first operation or a second operation depending on a state of said status bit;

generating an output of said first instruction in response to an input operand;

providing said output in response to performing said first operation; and

providing said input operand of said first instruction in response to performing said second operation.

8. A method for performing predicated instruction execution, comprising:

encoding a first instruction with a condition code that identifies a condition specified by a status bit in a status register;

executing a second instruction, wherein said executing includes setting said status bit in said status register;

setting a predicate flag in said status register responsive to said setting said status bit;

executing said first instruction, wherein said executing said first instruction includes:

decoding said condition code;

reading said predicate flag responsive to said decoding;

performing a first operation or a second operation depending on a state of said predicate flag;

generating an output of said first instruction in response to an input operand;

providing said output in response to performing said first operation; and

providing said input operand of said first instruction in response to performing said second operation.

9. The method of claim 8, wherein said setting a predicate flag comprises executing a third instruction which transfers a state of said status bit to said predicate flag.

10. The method of claim 8, wherein said setting a predicate flag comprises decoding a prefix in said second instruction, said prefix identifying a predicate flag to be set, responsive to said status bit being set.

11. The method of claim 8, wherein said encoding a first instruction with a condition code that identifies a condition specified by a status bit in a status register further comprises identifying a predicate flag in which said condition is recorded.

12. A microprocessor comprising:

a reservation station, wherein said reservation station is configured to receive an instruction, a first operand, and a second operand; and

a functional unit coupled to said reservation station, wherein said functional unit is configured to receive said instruction, said first operand, and said second operand from said reservation station, and wherein said functional unit comprises:

a functional execution unit configured to receive said instruction, said first operand, and said second operand from said reservation station, and wherein said functional execution unit is configured to provide a function output in response to

said instruction, said first operand, and said second operand; and

a predication unit configured to determine whether a predetermined condition has occurred, wherein said predication unit is configured to transmit said functional output if said predetermined condition has occurred, and wherein said predication unit is configured to transmit said first operand if said predetermined condition has not occurred.

13. The microprocessor of claim 12, further including a status register operably coupled to said reservation station and said functional unit, and wherein said predication unit is configured to read a bit in said status register indicative of whether said predetermined condition has occurred.

17. The microprocessor of claim 15, wherein said bit is a predicate flag bit in said EFLAGS register.

18. The microprocessor of claim 15, wherein said predicate flag bit in said EFLAGS register is set responsive to an evaluation of a status flag.

19. The microprocessor of claim 18, wherein said predicate flag is set by a SETcc command.

20. The microprocessor of claim 18, wherein said predicate flag is set by an instruction prefix that specifies a condition code.

First Hit Fwd Refs

Generate Collection Print

L10: Entry 14 of 18 File: USPT Aug 24, 1999

US-PAT-NO: 5943687
DOCUMENT-IDENTIFIER: US 5943687 A

TITLE: Penalty-based cache storage and replacement techniques

DATE-ISSUED: August 24, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Liedberg; N Per .ANG.	Tyreso			SE

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Telefonakiebolaget LM Ericsson	Stockholm			SE	03

APPL-NO: 08/ 818340 [PALM]
DATE FILED: March 14, 1997

INT-CL: [06] G06 F 13/18

US-CL-ISSUED: 711/156; 711/133, 711/135
US-CL-CURRENT: 711/156; 711/133, 711/135

FIELD-OF-SEARCH: 711/118, 711/133, 711/134, 711/135, 711/136, 711/156

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5546559	August 1996	Kyushima	711/133
☐	5555393	September 1996	Tanaka	711/133
☐	5649137	July 1997	Favor	395/383
☐	5671444	September 1997	Akkary	395/872
☐	5696932	December 1997	Smith	711/118
☐	5774685	June 1998	Dubey	395/381
☐	5787471	July 1998	Inoue	711/133

OTHER PUBLICATIONS

"PowerPC 604, RISC Microprocessor User's Manual", pp. 1-3 and 1-7, published by Motorola Inc. in Nov. 1994.
David A. Patterson, et al., "Computer Architecture a Quantitative Approach", Second Edition, Morgan Kaufmann Publishers, Inc., pp. 374-385, published in 1990.
William Johnson, "Superscaler Microprocessor Design", P T R Prentice-Hall Inc., 1991, pp. 18-21.
"Algorithm for Instruction Cache Arbitration Among Multiple Instruction Streams," IBM Technical Disclosure Bulletin, vol. 39, No. 8, Aug. 1996, pp. 113-116.
D. K. Chia, et al., "Replacement Algorithm using Priority Class Structure," IBM Technical Disclosure Bulletin, vol. 15, No. 12, May 1973, pp. 3803-3805.
Jong-Hong Bae, et al., "A Supplementary Scheme for Reducing Cache Access Time," vol. E79-D, No. 4, Apr. 1996, pp. 385-387.

ART-UNIT: 272

PRIMARY-EXAMINER: Cabeca; John W.

ASSISTANT-EXAMINER: Chow; Christopher S.

ATTY-AGENT-FIRM: Burns, Doane, Swecker & Mathis, L.L.P.

ABSTRACT:

Cache data replacement techniques enable improved performance in a computer system having a central processing unit (CPU), a cache memory and a main memory, wherein the cache memory has a plurality of data items stored therein. The cache data replacement techniques include associating a priority value with each of the stored data items, wherein for each data item, the priority value is an estimate of how much CPU stall time will occur if an attempt is made to retrieve the data item from the cache memory when the data item is not stored in the cache memory. When a cache entry must be replaced, the priority values are analyzed to determine a lowest priority value. One of the data items that has the lowest priority value is selected and replaced by a replacement data item. The priority value of a data item may be determined, as a function of how many other instructions have been fetched and stored in a buffer memory between a time interval defined by initiation and completion of retrieval of the data item from the main memory, wherein execution of the other instructions is dependent on completing retrieval of the data item. In other aspects of the invention, the priority values of cache entries may periodically be lowered in order to improve the cache hit ratio, and may also be reinitialized whenever the associated data item is accessed, in order to ensure retention of valuable data items in the data cache.

36 Claims, 7 Drawing figures

L10: Entry 14 of 18 File: USPT Aug 24, 1999

DOCUMENT-IDENTIFIER: US 5943687 A
TITLE: Penalty-based cache storage and replacement techniques

Detailed Description Text (7):
The exemplary CPU 101 includes instruction fetch logic 111 for determining the address of a next instruction and for fetching that instruction. The instruction set of the exemplary CPU 101 is designed to utilize operands that are stored in any of a number of addressable working registers 113. Therefore, before an instruction utilizing a source operand can be carried out, that operand must first be loaded into the working register from the data cache 105 by way of operand fetch logic 115, an arithmetic logic unit (ALU) 117, and a re-order buffer 119. Of course, the invention is by no means limited to use only in reduced instruction set computers (RISC) such as the one described here. To the contrary, it may easily be applied to complex instruction set computers (CISC) in which, among other differences, instructions are provided which allow the ALU to operate on data received directly from an addressed memory location without having to first load that data into a working register.

Detailed Description Text (8):
Together, the operand fetch logic 115 and the re-order buffer 119 provide out-of-order execution capability. The operand fetch logic 115 is responsible for assembling source operands which are to be applied to respective inputs of the ALU 117. To facilitate this task, the operand fetch logic 115 includes a number of reservation stations for storing either the source data itself or a sequence number (assigned to the as-yet-unexecuted instruction) which indicates that the source data has not yet been loaded into the working register 113.

Detailed Description Text (9):
The re-order buffer 119 is responsible for assembling destination data, and for keeping track of whether or not a given instruction has been executed, as well as dependencies between instructions. Thus, when an instruction is received, it is assigned a tag number and put into the re-order buffer 119. All previously stored instructions in the re-order buffer 119 are then examined to determine whether any of the sources of the new instruction match the destinations of any other of the previous instructions. If there is a match, then there is a dependency which prevents the new instruction from being immediately executed (i.e., the new instruction requires a source operand that has not yet been generated by an earlier instruction). As a result, the sequence number of the new instruction would be placed in the operand fetch logic 115 instead of the source data itself.

Detailed Description Text (10):
In this embodiment, the re-order buffer 119 has space for storing the destination data itself. Consequently, when the destination data becomes available, it is loaded into the re-order buffer 119, and an associated status flag is set to indicate that the data is valid. In order to speed up instruction execution when the destination data is also source data of a subsequent instruction, the destination data may be routed directly from the re-order buffer 119 to the appropriate reservation station in the operand fetch logic 115, rather than requiring that the data first be written to the working register 113 and then moving the data from the working register 113 to the operand fetch logic 115. To

speed instruction execution even further, the exemplary CPU 101 further includes a path from the output of the ALU 117 to the input of the operand fetch logic 115, in order to permit newly generated source data to be loaded even sooner.

Detailed Description Text (11):
The fact that generated data has been stored into the re-order buffer 119 is transparent to the executing program, which is only concerned with storing results into one of the working registers 113 or into the data cache 105 (or main memory 107). Consequently, the data must eventually be moved from the re-order buffer 119 into the destination indicated by the corresponding instruction. The act of moving generated data from the re-order buffer 119 to the appropriate destination is called "committing", "retiring" or "writing-back" the data. The architecture of the exemplary CPU 101 adopts the strategy of committing generated data only after all of the data associated with earlier instructions have been committed.

Detailed Description Text (16):
Type-3 data may be control data. Control data either directly or indirectly determines the operation of a conditional branch instruction. (Control data operates indirectly when it is used as an input variable in any number of calculation instructions for determining a resultant data item which is then used as the predicate for determining the operation of a conditional branch instruction.)

Detailed Description Text (32):
In view of the possibility for overestimating priority values under some circumstances, one might perform a more sophisticated analysis of the contents of the instruction buffers in the re-order buffer in order to count only those non-executed instructions that actually have a dependence on the subject data item. However, the overhead associated with this analysis makes it less attractive than the simple approach described above.

Detailed Description Text (35):
After these instructions are fetched, the contents of the re-order buffer 119 might look as in Table 1:

First Hit Fwd Refs

Generate Collection Print

L10: Entry 16 of 18 File: USPT Mar 9, 1999

US-PAT-NO: 5881308
DOCUMENT-IDENTIFIER: US 5881308 A

TITLE: Computer organization for multiple and out-of-order execution of condition code testing and setting instructions out-of-order

DATE-ISSUED: March 9, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Dwyer, III; Harry	Annandale	NJ		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 08/ 841549 [PALM]
DATE FILED: April 30, 1997

PARENT-CASE:
This application is a division of U.S. patent application Ser. No. 08/328,933 filed Oct. 25, 1994 now U.S. Pat. No. 5,630,157, which is a continuation of Ser. No. 07/721,853 filed Jun. 13, 1991 and now abandoned.

INT-CL: [06] G06 F 9/36

US-CL-ISSUED: 39/800.23; 395/391, 395/393, 395/566
US-CL-CURRENT: 712/23; 712/215, 712/217, 712/225

FIELD-OF-SEARCH: 395/800.23, 395/800.24, 395/390, 395/391, 395/392, 395/393, 395/566, 711/149, 711/125

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4293910	October 1981	Flusche et al.	
☐	4498136	February 1985	Sproul, III	
☐	4654785	March 1987	Nishiyama et al.	

☐	4722049	January 1988	Lahti	
☐	4760520	July 1988	Shintani et al.	
☐	4807115	February 1989	Horng	
☐	4833599	May 1989	Colwell et al.	
☐	4853840	August 1989	Shibuya	
☐	4901233	February 1990	Liptay	
☐	4903196	February 1990	Pomerene et al.	
☐	4942525	July 1990	Shintani et al.	
☐	4972342	November 1990	Davis et al.	
☐	4991080	February 1991	Emma et al.	
☐	4991090	February 1991	Emma et al.	
☐	4991133	February 1991	Davis et al.	
☐	5003462	March 1991	Blaner et al.	
☐	5072364	December 1991	Jardine et al.	
☐	5075844	December 1991	Jardine et al.	
☐	5142634	August 1992	Fite et al.	
☐	5185868	February 1993	Tran	
☐	5185871	February 1993	Frey et al.	
☐	5186697	February 1993	Johnson	
☐	5197132	March 1993	Steely, Jr. et al.	
☐	5226126	July 1993	McFarland et al.	
☐	5226130	July 1993	Favor et al.	
☐	5228131	July 1993	Ueda et al.	
☐	5257354	October 1993	Comfort et al.	
☐	5265213	November 1993	Weiser et al.	
☐	5283873	February 1994	Steely, Jr. et al.	
☐	5287467	February 1994	Blaner et al.	
☐	5363495	November 1994	Fry et al.	395/375
☐	5471593	November 1995	Brangin	
☐	5640588	June 1997	Vegesna et al.	395/800.23

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0301220	February 1989	EP	
0340453	November 1989	EP	
0378830	July 1990	EP	
0381471	August 1990	EP	

OTHER PUBLICATIONS

"Design of a Computer: The Control Data 6600"; J.E. Thornton; Scott, Foreman & Company; Glenview, IL; 1970; pp. 12-140.
"Detection and Parallel Execution of Independent Instructions"; Garold S. Tjaden et al; IEEE Transactions on Computers, vol. C-19, No. 10, Oct. 1970; pp. 889-895.
"The IBM System/360 Model 91: Storage System"; L.J. Boland et al.; IBM Journal, Jan. 1967; pp. 54-68.
"A Fast Instruction Dispatch Unit for Multiple and Out-of-Sequence Issuances"; H.C. Horng; Professor of Elec. Engineering; Stanford University; Nov. 1987.
"Limits on Multiple Instruction Issue"; Michael D. Smith et al.; Stanford University; ASPLOS-III Proceedings; Apr. 1989.
"Design Coices for the HPSm Microprocessor Chip"; Wen-mei Hwu et al.; Proceedings of the 20th Annual Hawaii International Conference on System Sciences; 1987; pp. 330-336.
"Reducing the Cost of Branches"; Scott McFarling et al.; Computer Systems Laboratory; Stanford University; 1986 IEEE; pp. 396-403.
"The ZS-1 Central Processor"; J.E., Smith et al.; Astronautics Corporation on America; Madison, Wisconsin; 1987 ACM; pp. 199-204.
"Designing a VAX for High Performance"; Tryggve Fossum et al.; 1990 IEEE; pp. 36-43.
"Pre-Decoding Mechanism for Superscalar Architecture"; Kenji Minagawa et al.; Toshiba Research and Development Center; Japan; 1991 IEEE; pp. 21-24.
"Fast Reset of Logical Facilities"; IBM Technical Disclosure Bulletin, vol. 31, No. 6, Nov. 1988; p. 275.
"Pipelined Register-Storage Architectures"; Steven R. Kunkel et al.; University of Wisconsin-Madison; 1986 IEEE; pp. 515-517.
"The Performance Potential of Multiple Functional Unit Processors"; A.R. Pleszkun et al.; University of Wisconsin-Madison; 1988 IEEe; pp. 37-43.
"An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors"; Ramon D. Acosta et al; 1986 IEEE; pp. 815-828.
Instruction Issue Logic for High-Performance, Interruptable Pipelined Processor; Gurindar S. Sohi et al.; University of Wisconsin-Madison; 1987 ACM.
"Implementing Precise Interrupts in Pipelined Processors"; James E. Smith; IEEE; vol. 37, No. 5, May 1988; pp. 562-573.
"The Metaflow Architecture"; Val Popescu et al.; IEEE Micro; 1991 IEEE; Jun. 1991; vol. 11, No. 3; pp. 10-73.
"The Performance Potential of Multiple Functional Unit Processors"; A.R. Pleszkun et al.; University of Wisconsin-Madison; 1988 IEEE; Computer Architecture News; vol. 16, No. 2; pp. 37-44.
"Building Parallelism Into the Instruction Pipeline"; Scott Chan et al.; Tandem Computers, Inc., Cupertiono, California; High Performance Systems; pp. 52-58.
"SIMP: A Novel High-Speed Single-Procesor Architecture"; Kazuaki Murakami et al.; Dept. of Information Systems; Kyushu University, Fukuoka, Japan; 1989 ACM; pp. 78-85.
"Available Instruction-Level Parallelism for Superscalar and Superpiplined Machines"; Norman P. Jouppi et al. Digital Equipment Corp., Western Research Lab; 1989 ACM; pp. 272-282.
"Instruction Schuduling for the IBM RISC System/6000 Processor"; H.S. Warren, Jr.; IBM J. Res. Develop., vol. 34, No. 1, Jan. 1990.
"Machine Organization of the IBM RISC System/6000 Processor"; G.F. Grohoski; IBM J. Res. Develop., vol. 34, NO. 1, Jan. 1990; pp. 37-58.
"Interrupt Handling for Out-of-Order Execution Processors"; H.C. Torng et al; School of Elec. Engineering; Phillips Hall, Cornell University, Ithaca, NY; pp. 1-16 (with Figures 1-9 included in addition to text pages).
"Limits on Multiple Instruction Issue"; Michael D. Smith et al.; Stanford University; Center for Integrated Systems; 1989 ACM; pp. 290-302.
"Boosting Beyond Static Scheduling in a Superscalar Processor"; Michael D. Smith et al.; Stanford University; Computer System Lab; 1990 IEEE; pp. 344-354.
"Dynamic Instruction Scheduling and the Astronautics ZS-1"; James E. Smith;

Astronautics Corp. of America; Jul. 1989 IEEE; pp. 21-35.
"Look-Ahead Processors"; Robert M. Keller; Dept. of Elec. Engineering, Princeton University; Computing Surveys, vol. 7, No. 4, Dec. 1975; pp. 177-195.
"Instruction Issue Logic in Pipelined Supercomputers"; Shlomo Weiss et al.; IEEE Transactions on Computers, vol. C-33, No. 11, Nov. 1984; pp. 1013-0122.
"An Instruction Issuing Mechanism for Performance Enhancements"H.C. Torng; Technical Report; Feb. 1984.
"Cache Memories"; Alan Jay Smith; Computing Surveys; vol. 14, No. 3, Sep. 1982; pp. 473-530.
"Dhrystone: A Synthetic Programming Benchmark"; Reinhold P. Weicker; Computing Practices; Oct. 1984, vol. 27, No. 10; pp. 1013-1030.
"VSLI Processor Architecture"; John L. Hennessy; IEEE Transactions on Computers; vol. C-33, No. 12, Dec. 1984; pp. 1221-1246.
"Concurrent VLSI Architectures"; Charles L. Seitz; IEEE Transactions on Computers, vol. C-33, No. 12, Dec. 1984; pp. 1246-1265.
"Critical Issues Regarding HPS, A High Performance Microarchitecture"; Yale N. Patt et al.; University of California, Berkley; 1985 ACM; pp. 109-116.
"An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors"; Ramon D. Acosta; IEEE Transactions on Computers; vol. C-35, No. 9, Sep. 1986; pp. 815-828.
"HPSm, a High Performance Restricted Data Flow Architecture Having Minimal Functionality"; Wen-mei Hwu et al.; University of California, Berkley; IEE 1986.
"New Computers for Artificial Intelligence Processing"; Benjamin W. Wah; University of Illinois at Urbana-Champaign; 1987 IEEE; pp. 10-15.
"Reducing the Branch Penalty in Pipelined Processors"; David J. Lilja; University of Illinois at Urbana-Champaign; 1988 IEEE; pp. 47-55.
"A VLIW Architecture for a Trace Scheduling Compiler"; Robert P. Colwell et al.; IEEE Transactions on Computers, vol. 37, No. 8, Aug. 1988; pp. 967-979.
"HPSm2: A Refined Single-Chip Microengine"; Wen-mei W. Hwu et al; University of Illinois, Urbana; 1988 IEEE; pp. 30-40.
"The Nonuniform Distribution of Instruction-Level and Machine Parallelism and Its Effect on Performance"; Norman P. Jouppi; 1989 IEEE; pp. 1645-1658.
Branch Prediction Strategies and Branch Target Buffer Design; Johnny K.F. Lee; Hewlett-Packard; Alan Jay Smith, University of California, Berkeley; Computer, vol. 17, No. 1; Jan. 1984.
"The 801 Minicomputer"; George Radin; IBM J. Res. Develop., vol. 27, No. 3, May 1983; pp. 237-246.
"An Efficient Algorithm for Exploiting Multiple Arithmetic Units"; R.M. Tomasulo; IBM; pp. 293-305.
"A VLSI RISC"; David A. Patterson et al., University of California, Berkeley; 1982 IEEE.
"Checkpoint Repair for High-Performance Out-of-Order Execution Machines"; Wen-Mei W. Hwu et al.; IEEE Transations on Computers, vol. C-36, No. 12; Dec. 1987; pp. 1497-1514.

ART-UNIT: 273

PRIMARY-EXAMINER: Donaghue; Larry D.

ATTY-AGENT-FIRM: Whitham, Curtis & Whitham Samodovitz, Esq.; Arthur J.

ABSTRACT:

Computer system with multiple, out-of-order, instruction issuing system suitable for superscalar processors with a RISC organization, also has a Fast Dispatch Stack (FDS), a dynamic instruction scheduling system that may issue multiple, out-of-order, instructions each cycle to functional units as dependencies allow. The basic issuing mechanism supports a short cycle time and its capabilities are augmented. Condition code dependent instructions issue in multiples and out-of-order. A fast

register renaming scheme is presented. An instruction squashing technique enables fast precise interrupts and branch prediction. Instructions preceding and following one or more predicted conditional branch instructions may issue out-of-order and concurrently. The effects of executed instructions following an incorrectly predicted branch instruction or an instruction that causes a precise interrupt are undone in one machine cycle.

15 Claims, 79 Drawing figures

L10: Entry 16 of 18 File: USPT Mar 9, 1999

DOCUMENT-IDENTIFIER: US 5881308 A
TITLE: Computer organization for multiple and out-of-order execution of condition code testing and setting instructions out-of-order

Detailed Description Text (79):
1. Instructions are issued to sets of reservation stations (holding buffers) 505, one set for each functional unit, where they wait until their operands become available. They are then issued to the attached functional unit for execution. If a reservation station is not available for an instruction, it stalls and blocks instructions behind it.

Detailed Description Text (80):
2. A register 510 may contain valid data or the name of a reservation station that will supply it with data in the future.

Detailed Description Text (81):
3. When an instruction is issued, the name of its reservation station is placed into the name field 515 of its destination register. Subsequent issues of instructions using that register as a source will take the name in the name field with them. They now know the name of the reservation station that will generate the data they need.

Detailed Description Text (82):
4. When an instruction completes, its result and reservation station name are broadcast. All instructions and registers waiting for that result can now identify and copy it.

Detailed Description Text (83):
The sequential issue of single instructions is an integral and necessary part of the algorithm. WAW and WAR hazards are eliminated because all instructions that read a register are issued before a subsequent instruction that writes the register. When an instruction writes to a register, all previously issued instructions that read it have copies of either the data or the reservation station's name that will produce the data. RAW hazards are also eliminated by the sequential nature of instruction issue.

Detailed Description Text (84):
It is interesting that Tomasulo's algorithm can also handle issues of multiple, in-order instructions containing no hazards (although this is not supported in the IBM 360/91). If enough reservation stations are available, the sequence could be issued. The (all different) result registers would not be sourced by other instructions within the sequence. Therefore, the reading of registers and the placing of reservation station names into them would occur correctly. All dependencies between instructions in different blocks would then be handled correctly by the algorithm. Some consider that Tomasulo's algorithm was used in the IBM System/360 model 195 and the IBM System/390.

Detailed Description Text (87):
HPSm is a minimum functionality implementation of High Performance Substrate (HPS) [HwPa 86, 87]. An HPSm instruction contains two operations that may have data

dependencies on each other. If they do, the instruction causes the hardware to forward the results of the first operation directly to the second. The execution mechanism examines sequential instructions and decomposes them into two operations, which, after register renaming, are integrated into data structures attached to functional units called node tables. Node tables operate much like Tomasulo's reservation stations. Instructions wait here for operands to be generated by the functional units and are then issued to functional units for execution. FIG. 6 is a block diagram of the HPSm machine.

Detailed Description Text (92):
Sohi improves the efficiency of Tomasulo's algorithm by concentrating all the instruction issuing logic and reservation stations into one mechanism called the Register Update Unit (RUU) [Sohi 90]. A reservation station is no longer permanently coupled to a particular functional unit and may be assigned as needed. As previously noted, Tomasulo's approach will stall if a reservation station is not available on the required functional unit. FIG. 7 shows a diagram of the RUU.

Detailed Description Text (100):
By centralizing data and control, the RUU enhances Tomasulo's approach with improved reservation station utilization and support for precise interrupts. Speedups are reported over serial issue of about 1.5 to 1.7, with one issue unit and bypass logic. Sohi claims that the RUU concept may be expanded to issue multiple instructions. With four issue units and bypass logic, speedups of 1.7 to 1.9 over serial issue are reported [PlSo 88].

Detailed Description Text (209):
The action of a Conditional Branch instruction, q.sub.CB, is predicated on a situation (condition) caused by the execution of one or more preceding instructions. Examples of conditions are an overflow out of a register and a result that is greater than zero. Conditional Branch instructions in different architectures test conditions using a variety of methods.

Detailed Description Text (212):
The data bandwidth of an execution unit is the maximum rate at which storage instructions can transfer data to and from memory. It may be limited to the maximum rate at which storage instructions can be executed (storage instruction throughput) or the maximum rate at which the memory system can store and fetch data (memory bandwidth). The data bandwidth of an execution unit with multiple FUs has to meet its data requirements; otherwise its throughput will have to be constrained. FUs may idle because the issuance and the execution of instructions dependent on the data are delayed. Previous proposals supporting out-of-order memory accesses schedule at most one memory request per cycle. Thornton's Scoreboard, Tomasulo's Reservation Station system, Sohi's RUU and Hwu and Pratt's HPSm issue at most one memory request per cycle (possibly out-of-order). The scheduling of storage instructions with Torng's Dispatch Stack has not been explored. This is one of the advances that I have made, and my improvements in this regard form part of my preferred embodiment. The SIMP mechanism initiates up to 4 read and 4 write requests per cycle to a Data Cache, but the scheduling algorithm of the storage instructions is not described.

Detailed Description Text (457):
The Future File system is shown in FIG. 58. Instructions are issued one-at-a-time and in-order to functional units. As an instruction is issued, information is placed in a queue called the Reorder Buffer 5805 and a stack called the Result Shift Register 5810. An instruction is issued if its operand registers are conflict free and it completes during a cycle in which no previous instruction completes. Functional units 5815 read operands from a set of working registers called the Future File 5820. When an instruction completes, a functional unit writes the result to the Future File and to the Reorder Buffer. The Reorder Buffer forwards the result to the architectural registers when preceding instructions have

completed. Results are written to the architected registers 5825 in instruction stream order.

Detailed Description Text (458):
The steps shown in FIG. 58 demonstrate the operation of the Future File. In step 1, instruction q.sub.i is issued to a functional unit and information associated with the issuance is written to the Reorder Buffer and Result Shift Register. The program counter and q.sub.i 's destination register are recorded in the Reorder Buffer slot pointed to by a tail pointer. The present value of the tail pointer. Pq, and the name of the functional unit assigned to q.sub.i are recorded in a Shift Result Register slot specified by q.sub.i 's execution time. For example, the third slot from the top is specified if an instruction requires 3 cycles to execute. If the slot is occupied, the issuance of q.sub.i is delayed. In this case, another instruction is scheduled to complete during the cycle that q.sub.i would have completed in.

Detailed Description Text (459):
Entries in the Shift Result Register shift one position toward its top position every cycle. Its top position, if not empty, contains a pointer to the Reorder Buffer slot with information about an instruction that just completed in a functional unit. The pointer associated with q.sub.i, P.sub.q, reaches the top position in the Result Shift Register as the functional unit executing q.sub.i produces a result. The result is transferred to the Future File and to the Reorder Buffer slot pointed to by P.sub.q, q.sub.i 's entry (step 3). If an interrupt occurred during q.sub.i 's execution, it is noted in q.sub.i 's entry at this time.

Detailed Description Text (460):
The entry pointed to by the head pointer of the Reorder Buffer is examined every cycle. If the entry contains a result and no exceptions are noted, the result is transferred to the Architectural File (step 4). If an exception is noted, the destination registers recorded in the Reorder Buffer are used to restore the Future File to the state it would have in a sequential machine just prior to the exception.

First Hit

End of Result Set

Generate Collection Print

L1: Entry 1 of 1 File: PGPB Jul 4, 2002

DOCUMENT-IDENTIFIER: US 20020087847 A1
TITLE: Method and apparatus for processing a predicated instruction using limited predicate slip

Detail Description Paragraph:
[0021] In principle, the predicate adds another input dependency to the first instruction. Unlike other data operands however, the predicate can only assume the "on" or "off" states. In general, an instruction in a processor can only be scheduled for execution after all of the instruction's inputs are resolved. In a simplistic implementation, a predicated instruction must be stalled until any predicates to the instruction are resolved.

Detail Description Paragraph:
[0022] A more sophisticated, in-order processor can issue a predicated instruction with an unresolved predicate. However, the predicated instruction must stall if the predicate is not resolved in the immediately following clock cycle. The stall prevents potentially incorrect data from being distributed by the bypass network. As soon as the predicate is resolved, the result of the instruction can either be distributed or discarded, as determined by the predicate, and the correctness of the data is guaranteed.

CLAIMS:

2. The method of claim 1, wherein dispatching the predicated instruction to an execution unit includes stalling the predicated instruction until all non-predicated dependencies are resolved.

8. A method of processing a predicated instruction comprising: receiving a predicated instruction in an execution stage of an in-order pipeline; stalling the predicated instruction until predicate is resolved; storing the result of the predicated instruction in a register, if the predicate is true; and deleting the result of the predicated instruction, if the predicate is not true.

14. A computer system comprising: a processor, wherein the processor includes: a plurality of in-order pipeline stages including at least one predicated instruction and a consumer instruction and wherein: the predicated instruction is received in an execution stage of the pipeline; if the predicated instruction is not followed by the consumer instruction in the next clock cycle then the predicated instruction is slipped to a previous stage in the pipeline; if the predicated instruction is followed by the consumer instruction in the next clock cycle then stalling the predicated instruction until predicate is resolved; and storing the result of the predicated instruction in a register, if the predicate is true; and deleting the result of the predicated instruction, if the predicate is not true. a system bus; a computer memory system; and an input/output device, wherein the system bus is coupled to the processor, the computer memory system and the input/output device.

First Hit

☐ Generate Collection | Print

L4: Entry 2 of 18 File: PGPB Jul 4, 2002

PGPUB-DOCUMENT-NUMBER: 20020087847
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020087847 A1

TITLE: Method and apparatus for processing a predicated instruction using limited predicate slip

PUBLICATION-DATE: July 4, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Kling, Ralph	Sunnyvale	CA	US	
Chamberlain, Jeffrey D.	Tracy	CA	US	
Wang, Perry H.	San Jose	CA	US	

APPL-NO: 09/ 751861 [PALM]
DATE FILED: December 30, 2000

INT-CL: [07] G06 F 9/00

US-CL-PUBLISHED: 712/233; 712/217
US-CL-CURRENT: 712/233; 712/217

REPRESENTATIVE-FIGURES: 3B

ABSTRACT:

A system and method of processing a predicated instruction is disclosed. A consumer instruction and a predicated instruction are received in an reservation station of an out-order processor. The consumer instruction depends on a result of the predicated instruction. The predicated instruction is dispatched to an execution unit for execution. The executed predicate instruction is stored in a re-order buffer.

First Hit

Generate Collection Print

L4: Entry 2 of 18 File: PGPB Jul 4, 2002

DOCUMENT-IDENTIFIER: US 20020087847 A1
TITLE: Method and apparatus for processing a predicated instruction using limited predicate slip

Brief Description of Drawings Paragraph:
[0008] FIG. 1B illustrates an in-order limited predicate slip CPU of one embodiment.

Brief Description of Drawings Paragraph:
[0013] FIGS. 3D-3E illustrate a limited predicate slip in-order CPU algorithm (with temporary result buffer) of one embodiment.

Brief Description of Drawings Paragraph:
[0015] FIGS. 5A-5C illustrate a limited predicate slip out-of-order CPU algorithm of one embodiment.

Detail Description Paragraph:
[0020] One property of a predicate implies that the predicated instruction can still be executed regardless of the value of the predicate since predicate does not alter the outcome of the computation. Instead, the predicate determines whether the outcome is to be used or not used. Predicate slip provides performance improvement by taking full advantage of this property of the predicate dependency.

Detail Description Paragraph:
[0024] One embodiment of an in-order pipeline with limited predicate slip is shown in FIG. 1B. The pipeline operates similar to the pipeline described above in FIG. 1A until the register read stage 162. In the register read stage 162 only source operand availability is checked. An unavailable guarding predicate does not stall execution. After execution, the result is written back to the register file 168, if the predicate is known and `true`. The result is discarded if the predicate is known and `false`. If the predicate is still unresolved the result is written back to a temporary result buffer 172 (associative buffer). This delays the update of the register file 168 until the predicate is resolved. The temporary result buffer 172 allows the pipeline to not stall due to an unavailable predicate in the cycle in which the register file 168 needs to be updated.

Detail Description Paragraph:
[0026] FIG. 1C shows a pipeline diagram of an out-of-order pipeline of one embodiment. The front-end 160 operates similar to that described in FIG. 1B in an in-order machine. The register read stage 162 obtains source operands from the register file 168, if available. The schedule/issue stage 174 decides when to execute an instruction. An instruction is executed when the scoreboard 170 indicates that the corresponding source operands are available. A conventional machine would regard the predicate as a source operand and prevent the schedule/issue stage 174 from issuing the instruction until the predicate is resolved and available. On embodiment of a limited slip pipeline ignores predicate availability at the schedule/issue stage 174. After execution, the results are written back to the reorder buffer 178. The reorder buffer 178 guarantees in-order updates of the architectural register file 168. In the limited predicate slip method, the reorder buffer 178 also assumes the functionality of the temporary

result buffer 172 that was added in the in-order pipeline described in FIG. 1B above.

Detail Description Paragraph:
[0028] It should be noted that the limited predicate slip method is not limited to the simple pipeline presented in the above example. More complex out-of-order pipelines that may contain register renaming, reservation stations and other enhancements are also contemplated. Separate reorder buffers and architectural register file are not required as long as the ordering of results becoming architecturally visible is maintained. Enhancements to the limited slip method depend on the pipeline structure and can also include early discarding of instructions with known `false` guarding predicates to save execution bandwidth.

Detail Description Paragraph:
[0030] If the predicate P1 is unavailable by the time instruction 4 is to be executed, the limited predicate slip algorithms, as described in more detail below, will delay a pipeline stall as long as possible. In the code example shown in FIG. 2 when instruction 5 needs to be executed, due to the dependence described above. Note that in real code there can be any number of unrelated instructions in between the ones shown (1-5). In summary, the following dependencies exist:

Detail Description Paragraph:
[0040] For one embodiment of a limited predicate slip, in-order CPU, with a temporary result buffer, the transition of an instruction from the register read stage to execution stage is substantially similar to that described above in FIG. 3B. FIG. 3D shows the transition of an instruction from the execution stage to write back stage in a limited predicate slip, in-order CPU, with a temporary result buffer. For each instruction in the execution stage, the availability of the source predicate of the instruction are checked in process block 320. Alternatively, the "oldest" instruction (i.e. the instruction that was issued to the execution stage first) is checked first and less aged instructions are checked in order of age. The scoreboard is queried to determine if the source predicate is ready or available. If the source predicate is not available, then the result of the instruction is written back to the temporary result buffer in process block 328. Alternatively, the temporary result buffer can be checked to determine if the temporary result buffer is full, as shown in process block 326, before the result is written back to the temporary result buffer. If the source predicate is available in process block 320, then the instruction can be advanced to the write back stage in process block 322 and the scoreboard is cleared in process block 324.

Detail Description Paragraph:
[0045] FIGS. 5A-5C illustrate a limited predicate slip out-of-order CPU of one embodiment. FIG. 5A illustrates the transition from issue to execution of one embodiment. For each instruction in the issue stage, the availability of the source operands of the instruction are checked in process block 500. Alternatively, the "oldest" instruction (i.e. the instruction that was issued to the register read stage first) is checked first and less aged instructions are checked in order of age. The scoreboard is queried to determine if the source operands are ready or available. If any one of the source operands are not available, then the pipeline is stalled until all of the source operands are available. If all of the source operands are available, then the instruction can be advanced to the execution stage in process block 502.

Detail Description Paragraph:
[0046] FIG. 5B shows the transition from execution to retiring the instruction in a limited predicate slip, out-of-order CPU of one embodiment. The source predicate is checked for availability in process block 510. If the predicate is available, then the result of the instruction is transferred to the reorder buffer in process block 514. Alternatively, the reorder buffer can also be checked to determine if the reorder buffer is full, before the result is transferred to the reorder buffer in

process block 512. Once the result of the instruction is transferred to the reorder buffer, the status entry for the instruction is cleared from the scoreboard in process block 516.

Detail Description Paragraph:
[0047] FIG. 5C shows one embodiment of processing the instruction in the reorder buffer in a limited predicate slip, out-of-order CPU. For each entry in reorder buffer, processed in order, the result of the instruction is checked for availability in process block 528. If the result is not ready, then the update of the reorder buffer is stalled until the result is ready. If the result is ready, the source predicate is checked for availability in process block 530. If the source predicate is not ready, then the update of the reorder buffer is stalled until the source predicate is ready. If the source predicate is ready, then the predicate whether the predicate is true or not is determined in process block 53. If the predicate is true, then the result of the instruction is written to the register file in block 534. The scoreboard is then cleared in process block 536 and the reorder buffer entry for the instruction is cleared in process block 538. If the predicate is false (not true) in process block 532, the result of the instruction is discarded in process block 540 and the scoreboard is then cleared in process block 536 and the reorder buffer entry for the instruction is cleared in process block 538.

CLAIMS:

9. The method of claim 8, further comprising: determining if a predicated instruction is followed by a consumer instruction in the next clock cycle, wherein the consumer instruction depends on a result of the predicated instruction; and slipping the predicated instruction to a previous stage in the pipeline if the predicated instruction is not followed by the consumer instruction in the next clock cycle.

14. A computer system comprising: a processor, wherein the processor includes: a plurality of in-order pipeline stages including at least one predicated instruction and a consumer instruction and wherein: the predicated instruction is received in an execution stage of the pipeline; if the predicated instruction is not followed by the consumer instruction in the next clock cycle then the predicated instruction is slipped to a previous stage in the pipeline; if the predicated instruction is followed by the consumer instruction in the next clock cycle then stalling the predicated instruction until predicate is resolved; and storing the result of the predicated instruction in a register, if the predicate is true; and deleting the result of the predicated instruction, if the predicate is not true. a system bus; a computer memory system; and an input/output device, wherein the system bus is coupled to the processor, the computer memory system and the input/output device.

First Hit Fwd Refs

Generate Collection Print

L5: Entry 4 of 44 File: USPT Sep 10, 2002

US-PAT-NO: 6449710
DOCUMENT-IDENTIFIER: US 6449710 B1

TITLE: Stitching parcels

DATE-ISSUED: September 10, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Isaman; David L.	San Diego	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
STMicroelectronics, Inc.	Carrollton	TX			02

APPL-NO: 09/ 429053 [PALM]
DATE FILED: October 29, 1999

INT-CL: [07] G06 F 9/38

US-CL-ISSUED: 712/216; 712/217, 712/218
US-CL-CURRENT: 712/216; 712/217, 712/218

FIELD-OF-SEARCH: 712/216, 712/217, 712/218, 712/226

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5129070	July 1992	Dorotte	
☐	5630149	May 1997	Bluhm	712/217
☐	5632023	May 1997	White et al.	712/218
☐	5737629	April 1998	Zuraski, Jr. et al.	712/23
☐	5751984	May 1998	Chang et al.	712/216
☐	5768556	June 1998	Canada et al.	712/216
☐	5911057	June 1999	Shiell	712/217
☐	5930521	July 1999	Tien et al.	712/23
	5937178	August 1999	Bluhm	712/218

☐				
☐	5983342	November 1999	Tran	712/218
☐	6047369	April 2000	Colwell et al.	712/217
☐	6108769	August 2000	Chinnakonda et al.	712/216

ART-UNIT: 2783

PRIMARY-EXAMINER: Treat; William M.

ATTY-AGENT-FIRM: Jorgenson; Lisa K. Munck; William A.

ABSTRACT:

The invention provides a method and system for performing instructions in a microprocessor having a set of registers, in which instructions which operate on portions of a register are recognized, and "stitching" instructions are inserted into the instruction stream to couple the instructions operating on the portions of the register. The "stitching" parcels are serialized along with other instruction parcels, so that instructions which read from or write to portions of a register can proceed independently and out of their original order, while maintaining the results of that out-or-order operation to be the same as if all instructions were performed in the original order. In a preferred embodiment, the choice of stitching parcels is optimized to the Intel x86 architecture and instruction set.

25 Claims, 10 Drawing figures

L5: Entry 4 of 44 File: USPT Sep 10, 2002

DOCUMENT-IDENTIFIER: US 6449710 B1
TITLE: Stitching parcels

Drawing Description Text (6):
FIG. 5 is a block diagram that shows the details of the register renaming scoreboard.

Detailed Description Text (35):
A system 400 includes an instruction cache 401, an instruction fetch buffer 402, an instruction parse logic 403, an instruction decode FIFO 404, four multiplexors 405, a multiplexor 406, an instruction issue FIFO 407, a stitch parcel FIFO 408, a multiplexor 409, an issue control logic 410, an instruction shelf 411, a result shelf 412, a register file 413, an execution unit 414, a switch 415 and a register renaming scoreboard 420.

Detailed Description Text (38):
The output of the instruction decode FIFO 404 is routed to the register renaming scoreboard 420 via the multiplexors 405 and 406. Absent the need to insert stitching parcels, the output of the register renaming scoreboard 420 is routed to the instruction issue FIFO 407 via a switch 415. If there is a need to insert a stitching parcel contained in the register renaming scoreboard 420, switch 415 will also route the output from the register renaming scoreboard 420 to the stitch parcel FIFO 408.

Detailed Description Text (39):
Major functions of the register renaming scoreboard 420 include the following activities: first, the register renaming scoreboard 420 allocates and assigns a sequential parcel identifier to parcels that do not require stitching. It does not assign an identifier to any parcel that requires stitching or to any parcel that follows a parcel that requires stitching. In the preferred embodiment, the identifier allocated and assigned by the register renaming scoreboard 420 includes a unique sequence of seven bits. Retirement logic (not shown) retires parcels in the original program order. The original program order is ascertained by looking to the identifier assigned to each parcel by the register renaming scoreboard 420. After a parcel has been retired, the parcel identifier is used to read that execution result from result shelf 412 and write the result to register file 413. This informs the register renaming scoreboard 420 that the identifier is free to be reallocated. Since the result shelf 412 is a 64 word RAM, the register renaming scoreboard 420 guarantees that no more than 64 identifiers can be assigned to parcels at any one time.

Detailed Description Text (40):
Secondly, the register renaming scoreboard 420 records the register specifier 110 of the target register 103 (if any) of each parcel to which it assigns an identifier. The target register specifier is determined from the x86 instruction by the parse logic 403; the target identifier is written into the instruction decode FIFO 404. The register renaming scoreboard 420 can tell if the parcel wrote only a portion of the target 32 bit register by looking at the target register specifier. This register specifier is the internal representation of each instruction parcel's operand registers 102 and destination register; it is commonly included in

multiprocessors available from Intel, AMD and companies.

Detailed Description Text (41):
Thirdly, the register renaming scoreboard 420 determines which unretired parcel most recently wrote to each portion of the operand register 102. The register renaming scoreboard 420 makes this determination by looking to the register specifier of the target register 103 of every instruction 100. Thus, if the register renaming scoreboard 420 determines that no unretired parcel wrote to those portions of the register, then the operand value will be found in its entirety in the register file 413 if some portions of operand register 102 are needed. If two or more portions of the operand register 102 are needed and the register renaming scoreboard 420 determines that those values were not all the result of a single earlier unretired parcel, then a stitch parcel(s) is needed to get the correct value of that operand register 102.

Detailed Description Text (42):
If stitch parcels are not required, the output from the register renaming scoreboard 420 includes an indication of whether the last parcel that writes to the operand register 102 has been retired. If the last parcel that wrote to a given portion of the operand register 102 (termed the "locker" of that portion) has not been retired, then the output will include the identifier of that locker. The locker identifier of a portion of an operand register 102 is identical to both the identifier of the parcel (assigned by the register renaming scoreboard 420 in the manner described above) and the address of the result shelf 412 location in which the result is stored from the time it is computed until such time when the locker parcel is retired.

Detailed Description Text (43):
Issue control logic 410 examines the contents of instruction issue FIFO 407 and determines if any stitch parcels require insertion. As detailed above, switch 415 wrote the output of the register renaming scoreboard 420 to instruction issue FIFO 407. For each parcel that does not need stitching, issue control logic 410 causes multiplexor 409 to route that parcel directly from the instruction issue FIFO 407 to the instruction shelf 411. Instruction shelf 411 receives all parcels from either the instruction issue FIFO 407 or the stitch parcel FIFO 408. The instruction shelf 411 implements the out-of-order execution sequencing of the parcels contained in it. A parcel is ready for execution whenever the instruction shelf 411 determines that the complete value of each operand register 102 is available.

Detailed Description Text (48):
The first step of this process begins when issue control logic 410 causes multiplexors 405 and 406 to route input from the instruction issue FIFO 407 to the register renaming scoreboard 420. Issue control logic 410 also causes switch 415 to write the output of the register renaming scoreboard 420 into the stitch parcel FIFO 408 instead of instruction issue FIFO 407. The contents of issue instruction FIFO 407 are unchanged by this first step. Issue control logic also prevents any new parcels from being written into the instruction shelf 411.

Detailed Description Text (49):
The second step of this process also involves issue control 410. Issue control logic 410 causes multiplexors 406 and 405 to route the output from instruction issue FIFO 407 back to the input of register renaming scoreboard 420. Issue control logic 410 causes switch 415 to write the outputs of the register renaming scoreboard 420 to the inputs of instruction issue FIFO 407. Lastly, issue control logic 410 causes multiplexor 409 to route the output of stitch parcel FIFO 408 to the instruction shelf 411 and allows those parcels to be written into the instruction shelf.

Detailed Description Text (50):

The entire process ends when issue control logic 410 causes multiplexors 405 and 409 to return to their original positions. Issue control logic 410 causes multiplexor 405 to take input of register renaming scoreboard 420 from the instruction decode FIFO 404. Multiplexor 409 continues to take all input to the instruction shelf 411 from the stitch parcel FIFO 408. As soon as stitch parcel FIFO 408 is empty, issue control logic 410 causes multiplexor 409 to take the instruction shelf 411 input from instruction issue FIFO 407. Once multiplexors 405 and 409 have returned to these original positions, the system 400 is complete and the insertion of stitch parcels has been accomplished.

Detailed Description Text (51):
The Structure of the Register Renaming Scoreboard

Detailed Description Text (52):
FIG. 5 is a block diagram that shows the details of the register renaming scoreboard.

Detailed Description Text (53):
The register renaming scoreboard 420 includes an identifier generator 501, four parcel identifier wires 502a-502d, four register specifier wires 503a-d, a decoder 504, 64 sets of four decoder output signals 505a-505d, an array of 64 identical locker circuits 506, a set of 64 wires 507, a set of 64 register specifier wires 508, a state block 510, eight identical operand blocks 511a-h (not all shown) and eight register specifier wires 512a-h (not all shown). Each operand block 511 has six outputs 513, 514, 515, 516, 517 and 518.

Detailed Description Text (68):
If the extended field of selected register specifier 605 is 0, then either no parcel having identifier i is being routed to the register renaming scoreboard 420 for that particular cycle or such parcel does not write the extended portion of a target register 103 (not all parcels write to a register). In this case, the youngest locker of the extended portion will not be forced to 1, but it could still be cleared to 0 by the locker clear circuit 630. The locker clear circuit inputs the register number 640, the target register 103 specifiers 503a-d of all four parcels that are inputs to the register renaming scoreboard 420 and a unique one of the 64 locker clear signals 507.

Detailed Description Text (72):
Each locker clear circuit 630 inputs a unique one of the 64 retire signals 507. Input 507 to locker circuit i, i=0 . . . 63, is 1 when retirement logic (not shown) determines that the parcel with identifier i is retiring. Since the register renaming scoreboard 420 stores information only about unretired lockers, all three NOR gates 705a-c unconditionally output a 0 when retire signal 507 is 1. This clears the three youngest locker registers 620a-c.

Detailed Description Text (73):
Each of the identical target clear blocks 701a-d in the locker clear circuit 630 included in a locker circuit 506 inputs both (1) the register specifier register number 640 that is stored in that locker clear circuit 506 and (2) the four target register specifiers 503a-d that are input to the register renaming scoreboard 420. Output 702a of the target clear block 701a is 1 if the first instruction at the output of multiplexor 405 writes to the extended portion of the register whose number 640 is stored in this locker circuit. Similarly, outputs 703a and 704a are 1 if the first instruction at the output of multiplexor 405 writes to the high or low portion, respectively of that target register 103. Outputs 702b-d, 703b-d and 704b-d are 1 if the corresponding second, third or fourth instructions at the output of the multiplexor 405 write to the extended, high or low portion, respectively, of that register.

Detailed Description Text (74):

The record contained in the register renaming scoreboard 420 of the most recent instruction to write to the extended portion of a register is erased if any of the four instructions at the output of multiplexor 405 write to the extended portion of that register. This happens when a more recent instruction (specifically, the output of multiplexor 405) writes to the same portion of that register. Under such circumstances, at least one of the outputs from the target clear blocks will be a 1, forcing the output 631a of NOR gate 705a to 0. This erases any 1 from register 620a. Similar reasoning applies to the high and low register portions, which are cleared by the outputs 631b and 631c of NOR gates 705b and 705c respectively.

Your search matched **1** of **1013964** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance** in **Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or entering a new one in the text box.

Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

1 **Phi-predication for light-weight if-conversion**
Weihaw Chuang; Calder, B.; Ferrante, J.;
Code Generation and Optimization, 2003. CGO 2003.
International Symposium on , 23-26 March 2003
Pages:179 - 190

[Abstract] [PDF Full-Text (386 KB)] **IEEE CNF**

PORTAL Subscribe (Full Service) Register (Limited Service, Free) Login

Search: ◉ The ACM Digital Library ○ The Guide

THE ACM DIGITAL LIBRARY

Enter words, phrases or names below. Surround phrases or full names with double quotation marks.

Desired Results:

must have **all** of the words or phrases
stall predicate

must have **any** of the words or phrases

must have **none** of the words or phrases

Name or
Author
Edited
Review

Only search in:*
○ Title ○ Abstract ○ Review ◉ All Information

*Searches will be performed on all available information, including full text where available, unless specifi

ISBN / ISSN: ◉ Exact ○ Expand

DOI: ◉ E

Published:

By: ◉ all ○ any ○ none

In: ◉ all ○ any ○ none

Since: Month Year

Before: Month Year

As: Any type of publication

Confere
Sponsor
Conferer
Conferer

Classification: (CCS) ☐ **Primary Only**

Classified as: ◉ all ○ any ○ none

Results
☐ Full

Subject Descriptor: ◉ all ○ any ○ none

Keyword Assigned: ◉ all ○ any ○ none

The ACM Portal is published by the Association for Computing

Terms of Usage Privacy Policy Code o

Results 1 - 6 of 6

1 A survey of processors with explicit multithreading
Theo Ungerer, Borut Robi?, Jurij Šilc
March 2003 ACM Computing Surveys (CSUR), Volume 35 Issue 1
Full text available: pdf(920.16 KB) Additional Information: full citation, abstract, references,

Hardware multithreading is becoming a generally applied technique in the next ge multithreaded processors are announced by industry or already into production in microprocessors, media, and network processors.A multithreaded processor is abl control in parallel within the processor pipeline. The contexts of two or more threa separate on-chip register sets. Unused i ...

Keywords: Blocked multithreading, interleaved multithreading, simultaneous mult

2 A dynamic scheduling logic for exploiting multiple functional units in single cl
Prasad N. Golla, Eric C. Lin
February 1999 Proceedings of the 1999 ACM symposium on Applied computing
Full text available: pdf(1.19 MB) Additional Information: full citation, references, index t

Keywords: Tomasulo's algorithm, computer architecture, microprocessor, multithr

3 A framework for expressing and combining preferences
Rakesh Agrawal, Edward L. Wimmers
May 2000 ACM SIGMOD Record , Proceedings of the 2000 ACM SIGMOD internationa Volume 29 Issue 2
Full text available: pdf(778.31 KB) Additional Information: full citation, abstract, references, citi

The advent of the World Wide Web has created an explosion in th As the range of potential choices expand, the time and effort requ expands. We propose a formal framework for expressing and com address this problem. Preferences can be used to focus search qu results. A preference is expressed by the user for an entity which fields; each field can t ...

4 Database issues for event-based middleware: Relational subscription middle publish-subscribe
Yuhui Jin, Rob Strom
June 2003 Proceedings of the 2nd international workshop on Distributed event-ba
Full text available: pdf(103.95 KB) Additional Information: full citation, abstract, references

We present a design of a distributed publish-subscribe system that extends the fu with "relational subscriptions", to support timely updates to state derived from pu high throughput, scalability, and reliability.Critical to our design is our service gua Eventual correctness is a weaker guarantee than the ACID properties of conventic deliver state that is "jus ...

Keywords: continuous queries, event distribution systems, monotonicity, relationa

5 Internet Nuggets
Mark Thorson
June 2000 ACM SIGARCH Computer Architecture News, Volume 28 Issue 3
Full text available: pdf(428.68 KB) Additional Information: full citation, ab

This column consists of selected traffic from the *comp.arch* newsgroup, a forum fc on Internet--- an international computer network.

6 Retrospective: instruction issue logic for high-performance, interruptable pip
Gurindar S. Sohi
August 1998 25 years of the international symposia on Computer architecture (selec
Full text available: pdf(342.89 KB) Additional Information: full citation, references, index terms

Results 1 - 6 of 6

Your search matched **0** of **1013964** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance** in **Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or entering a new one in the text box.

Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

Results:
No documents matched your query.

Your search matched **0** of **1013964** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance** in **Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or entering a new one in the text box.

Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

Results:
No documents matched your query.

Your search matched **0** of **1013964** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance** in **Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or entering a new one in the text box.

Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

Results:
No documents matched your query.

Your search matched **0** of **1013964** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance** in **Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or entering a new one in the text box.

Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

Results:
No documents matched your query.

First Hit

End of Result Set

Generate Collection | Print

L1: Entry 1 of 1 File: PGPB Jul 4, 2002

PGPUB-DOCUMENT-NUMBER: 20020087847
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20020087847 A1

TITLE: Method and apparatus for processing a predicated instruction using limited predicate slip

PUBLICATION-DATE: July 4, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Kling, Ralph	Sunnyvale	CA	US	
Chamberlain, Jeffrey D.	Tracy	CA	US	
Wang, Perry H.	San Jose	CA	US	

APPL-NO: 09/ 751861 [PALM]
DATE FILED: December 30, 2000

INT-CL: [07] G06 F 9/00

US-CL-PUBLISHED: 712/233; 712/217
US-CL-CURRENT: 712/233; 712/217

REPRESENTATIVE-FIGURES: 3B

ABSTRACT:

A system and method of processing a predicated instruction is disclosed. A consumer instruction and a predicated instruction are received in an reservation station of an out-order processor. The consumer instruction depends on a result of the predicated instruction. The predicated instruction is dispatched to an execution unit for execution. The executed predicate instruction is stored in a re-order buffer.

Your search matched **0** of **1013964** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance** in **Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or entering a new one in the text box.

Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

Results:
No documents matched your query.